Proyecto Fin de Máster en Ingeniería de Computadores
Curso 2006-07

Entorno de Desarrollo para Ubicación de Tareas en Multitarea Hardware 2D

Miguel Angel García de Dios

Dirigido por:
Hortensia Mecha López
Dpto. Arquitectura de Computadores y Automática

Máster en Investigación Informática
Facultad de Informática
Universidad Complutense de Madrid

Índice

1.-Introducción.

-Propósito.

-En esta memoria presentamlos el desarrollo de un entorno de trabajo para estudiar distintas técnicas de ubicación de tareas en un sistema de hardware dinámicamente reconfigurable como es el de una FPGA [1] [2] [3]. También de permitir el estudio de distintas medidas de fragmentación para un estado concreto de la FPGA.

-Para ello se ha desarrollado una aplicación en Java formada por un interfaz gráfico en el cual podemos crear cualquier estado de ejecución de una FPGA del tamaño deseado y visualizar el valor de la fragmentación a través de distintas medidas. Podemos crear una tarea y ver dónde iría colocada esa tarea en función de la técnica de ubicación escogida. Y además podemos ejecutar un archivo compuesto por un lote de tareas y obtener de forma rápida datos relevantes de la ejecución de ese lote de tareas en la FPGA.

-Multitarea Hardware

-Uno de los conceptos base de este trabajo es el de **multitarea hardware**, es decir, la ejecución simultánea de varias tareas HW en el mismo dispositivo como es el de una FPGA. Algunos de los argumentos que justifican abordar este concepto son los siguientes:

-El **aumento continuado de tamaño de los dispositivos reconfigurables**, que incluso aunque lleve al agotamiento de las posibilidades de la tecnología CMOS puede continuar mediante los dispositivos basados en tecnología molecular. La organización en 2D de los dispositivos parece ser además una característica apropiada para esta nueva tecnología.

-La aparición de la posibilidad de **reconfiguración parcial**, que permite cargar una tarea sin interrumpir el funcionamiento de las demás.

-La **reducción de los tiempos de reconfiguración**, tanto tecnológica como arquitectónicamente, mediante el uso, por ejemplo, de caches de configuraciones o de múltiples planos de configuración (contextos).

-Necesidad de ubicación de tareas.

-El hardware dinámicamente reconfigurable y más en concreto las FPGAs, tiene una gran utilidad entre los trabajos que tienen restricciones en tiempo real. Estas restricciones de tiempo real se traduce a que las tareas que van a ejecutarse en la FPGA tienen asociado un tiempo máximo de finalización. Por lo tanto, si tenemos una serie de tareas que van llegando a la FPGA y tienen que ser ejecutadas, con un tiempo máximo de finalización, nos lleva a pensar que el cómo ubiquemos esas tareas en el área libre disponible de la FPGA, mejorará notablemente el rendimiento ya que podremos ejecutar mayor número de tareas.

-De esta idea surge la necesidad de aplicar técnicas de ubicación a las tareas que llegan a ejecutarse en la FPGA ya que cuanto mejor se coloquen las tareas en el área libre, más tareas podrán ejecutarse en la FPGA.

-Pero para poder aplicar distintas técnicas de ubicación, es necesario tener previamente una estructura para la gestión y almacenamiento del área libre de la FPGA, para así poder aplicar esas técnicas de ubicación sobre esas estructuras en concreto.

-Fragmentación.

- Se entiende que la distribución del área libre de una FPGA es mejor o peor si el área está más o menos fragmentada.

-La fragmentación [5] significa subdivisón o separación de algo. En nuestro caso la fragmentación tiene más que ver con lo más o menos subdividido que esté el área libre de la FPGA. Pero no hay una única forma de medir la fragmentación, de hecho existen distintas medidas cada una basada en unas idea u otras.

2.- Estructuras de información para la gestión del área libre

-Antes de nada hay que aclarar que el área de una FPGA se representa como una matriz de N x M celdas en la cual se irán ubicando tareas siempre con formas rectangulares. En la *figura 1* se puede observar un ejemplo de una representación del área de una FPGA totalmente vacía de tamaño 20 de ancho por 20 de alto. En este caso el área de la matriz es cuadrada, pero puede ser rectangular.

-Para poder gestionar el área libre en una FPGA, es necesario soportar, mantener y actualizar la información necesaria, pero a su vez, necesitamos que tal estructura nos facilite de forma casi inmediata, información sobre dónde podemos ubicar una nueva tarea.

-Se proponen dos estructuras de información para la gestión del área libre de una FPGA, y son las siguientes:

-A través de **MERs.**
-A través de **listas de vértices.**

-Cada estructura es muy distinta de la otra y, aunque ambas almacenan el área libre de nuestro dispositivo, cada una tiene sus ventajas y sus inconvenientes.

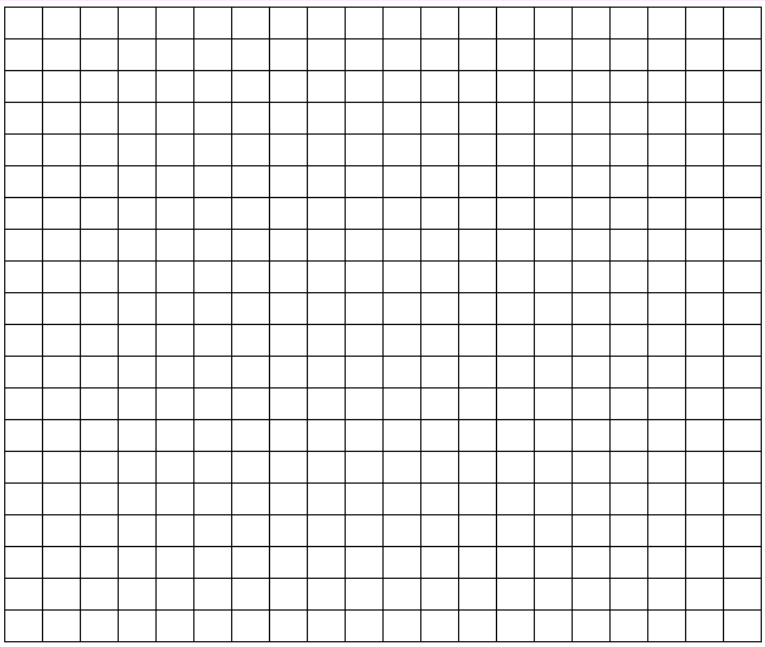


Figura 1: FPGA de tamaño 20x20 vacía.

-Según vayan llegando tareas a ejecución, se irán colocando en la FPGA ocupando parte del área libre disponible tal y como muestra la *figura 2*. En dónde se coloquen las tareas a ejecutar en el área libre depende de la heurística de colocación que usemos, pero la complejidad de la heurística de selección que usemos depende a su vez de la estructura que tengamos para almacenar el área libre, ya que según la estructura elegida (MERs o listas de vértices) resultará más o menos complejo aplicar una u otra heurística. De ahí la importancia que tiene el almacenamiento del área libre, ya que en función de cómo lo hagamos, tendremos unos resultados u otros en el rendimiento de una serie de tareas con unas restricciones determinadas.

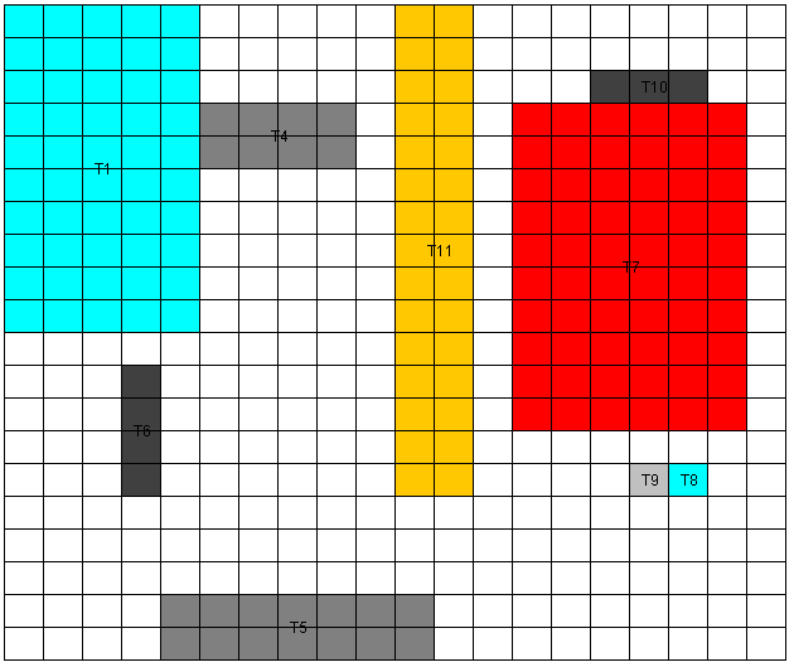


Figura 2: FPGA de tamaño 20x20 con una serie de tareas en ejecución.

-En un hardware dinámicamente reconfigurable como es una FPGA, la gestión del área libre es vital para conseguir un buen rendimiento del dispositivo a lo largo de la ejecución de numerosas tareas. Una buena gestión permitirá ejecutar mayor número de tareas y en menos tiempo.

-A continuación se pasa a describir cada una de las técnicas de almacenamiento del área libre de la FPGA.

2.1 MERs

-Una forma de tener almacenada el área libre de la FPGA podría ser un conjunto de rectángulos donde cada rectángulo representa una parte del área libre. Ya que la FPGA tiene forma rectangular y las tareas son también de forma rectangular, entonces el área libre siempre puede venir dada como un conjunto de rectángulos.

-La palabra MER [8] significa “Maximun Empty Rectangle”, y lo que quiere decir es rectágulo formado por el máximo área libre posible. En la *figura 3a*, podemos ver un ejemplo de un MER para un estado concreto de una FPGA y en la *figura 3b* vemos para otro estado concreto de una FPGA, cual sería el conjunto de todos sus MERs.

-Con esta definición de MER, para cualquier estado de una FPGA, si calculamos todos sus MERs obtendremos un conjunto de rectángulos mínimo, los cuales formarían el

total del área libre de nuestra FPGA con la peculiaridad de que ningún MER contendría por completo a otro MER por definición propia. El conjunto de MERs es el conjunto de rectángulos con la mínima cardinalidad, que componen todo el área libre de una FPGA.

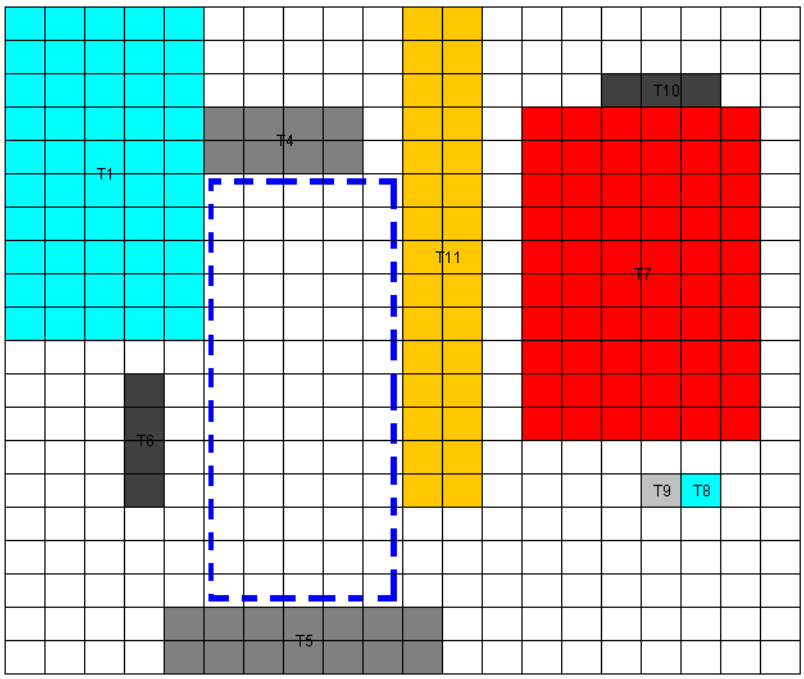


Figura 3a: de azul oscuro y punteado, uno de los varios MERs para esta situación.

-A partir de una situación concreta de una FPGA con una serie de tareas ejecutándose, definimos **evento** como la acción de que entre una nueva tarea en ejecución en la FPGA o la finalización de ejecución de una tarea y su correspondiente salida de la FPGA.

-Sin entrar en detalles de los algoritmos necesarios para el cálculo de MERs [7] para una situación en concreto del área libre de una FPGA, si es conveniente resaltar que cada vez que ocurre un evento, es necesario recalcular el conjunto de MERs ya que la inclusión o exclusión de una tarea en la FPGA causa que el conjunto de MERs cambie totalmente y por lo tanto tengan que ser recalculados lo que implica una gran penalización en el rendimiento partiendo de la base de que el cálculo de MERs tiene una complejidad de N^3.

-Como se ha dicho anteriormente, el usar como estructura de almacenamiento del área libre los MERs afecta al rendimiento de algunas técnicas de colocación de nuevas tareas que van a ejecutarse en la FPGA. Más en concreto, las heurísticas de adyacencia 2D y 3D y las basadas en la fragmentación, empeoran el rendimiento si la estructura de datos usada son los MERs ya que el proceso de colocación de la tarea dentro de un posible MER a partir de estas técnicas es mucho más costoso que en la estructura de datos basada en vértices, ya que en ésta, la colocación de la nueva tarea viene dada ya por un vértice determinado.

-Las técnicas de colocación de nuevas tareas en la FPGA alicadas conjunto de MERs parten de la base de que cada tarea irá colocada dentro de un MER y qué MER será el mejor o en qué posición dentro del MER lo decide ya cada tipo de técnica.

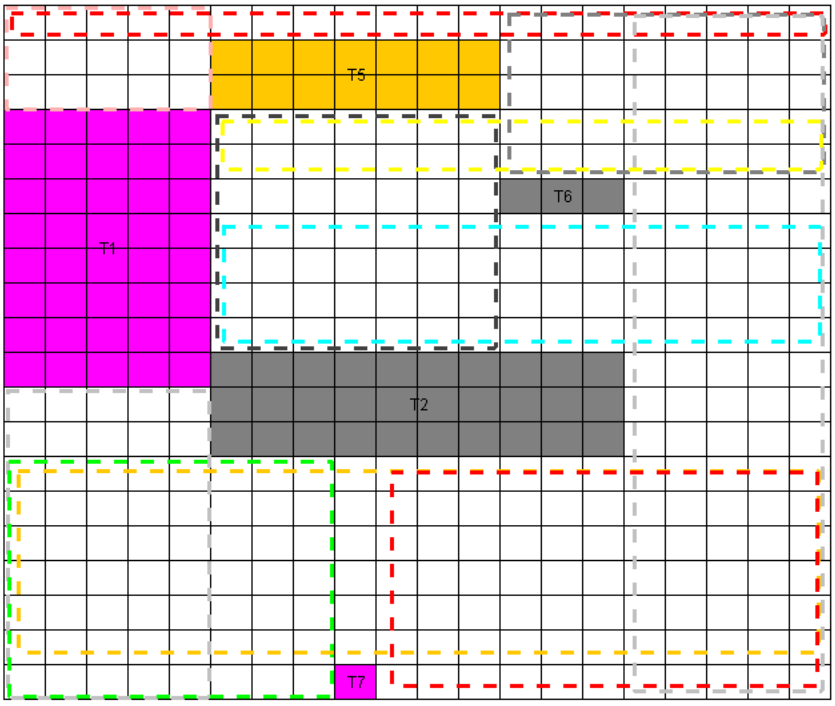


Figura 3b: conjunto de MERs para este estado concreto de una FPGA.

-Para finalizar, se muestra a continuación un ejemplo de una situación de una FPGA con su conjunto de MERs y a continuación añadiremos una tarea y vemos como ese conjunto de MERs cambia. En la *figura 3c* podemos observar el conjunto de MERs para una situación de la FPGA en donde solamente hay una tarea.

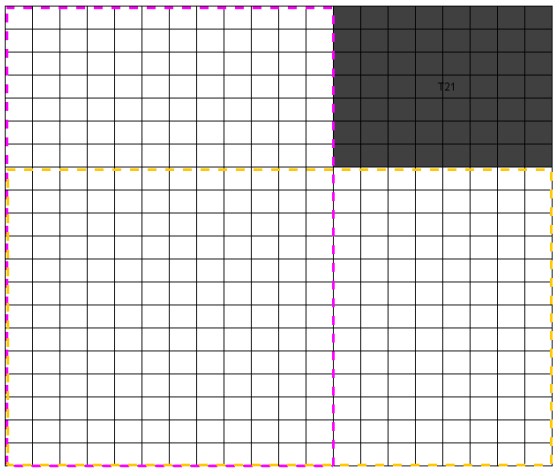


Figura 3c.

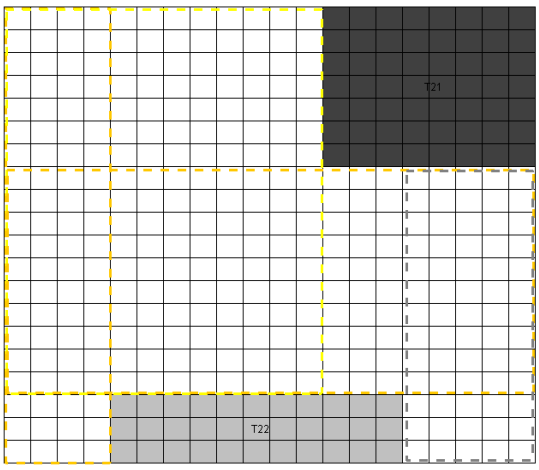


Figura 3d.

Ahora si añadimos una tarea más a ejecución en la FPGA, podemos observar en la *figura 3d* cómo cambia tanto el número como la forma de los MERs ya que éstos han tenido que ser recalculados.

2.2 Listas de vértices.

-Antes de explicar la estructura formada por listas de vértices [6], daremos unas definiciones necesarias para su comprensión:

-Definimos **vértice** como una esquina o un rincón producido en el área libre de la FPGA debido a la propia FPGA o al conjunto de tareas que se están ejecutando en ella.

-**Vértice candidato**: es aquel vértice que es una esquina. Se le llama candidato porque en los vértices que son esquina son los que nos van a interesar como candidato a incluir en él una nueva tarea. En la *figura 4* podemos ver un ejemplo de un vértice candidato y de un vértice no candidato.

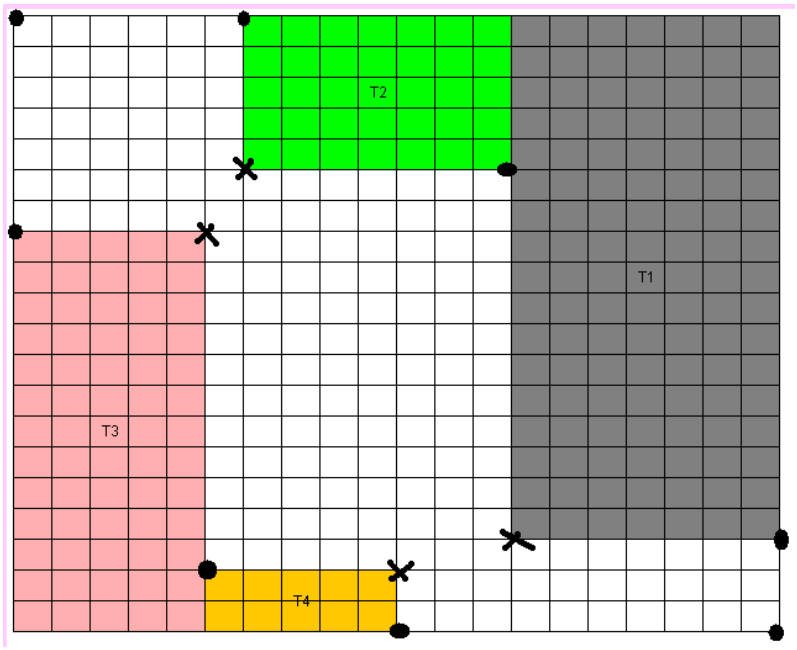


Figura 4: los vértices marcados con círculos son candidatos y los marcados con aspas son no candidatos.

-**Hueco**: es la zona de área libre que queda delimitada o por la propia FPGA o por tareas que están en ejecución. En la *figura 5* podemos ver un ejemplo de una situación de la FPGA con huecos, en concreto tres huecos.

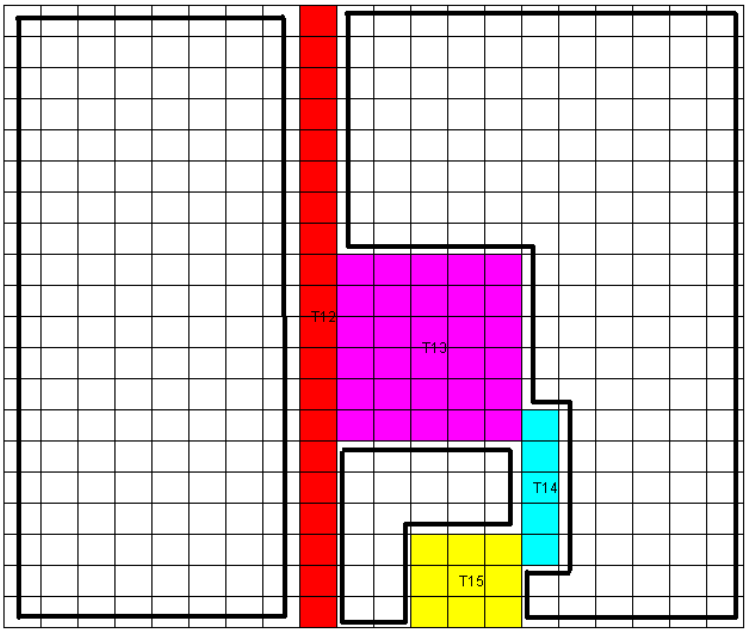


Figura 5: FPGA en la cual, aparecen tres huecos bordeados de negro.

-**Isla**: es la tarea o conjunto de tareas rodeadas por todas partes de área libre.

-**Vértice virtual**: es el vértice que sin ser ni una esquina ni un hueco, sin embargo es necesario porque engancha a la lista de vértices, los vértices pertenecientes a una isla. En la *figura6* podemos observar una situación de la FPGA en la que aparecen dos islas: una formada por una sola tarea y la otra formada por varias tareas, ambas con sus correspondientes vértices virtuales.

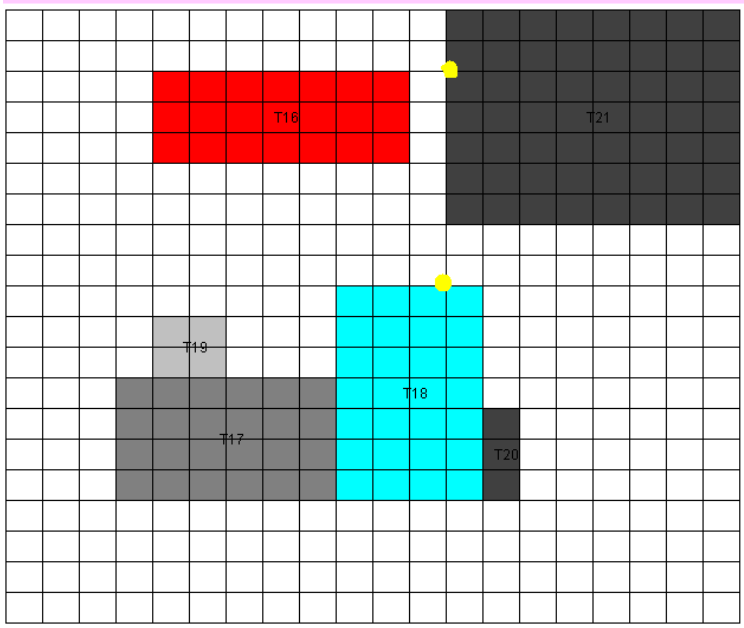


Figura 6: T16 forma una isla al igual que el conjunto de tareas T17, T18, T19 y T20 que forman la otra isla. Los puntos amarillos muestran los vértices virtuales.

-Ahora a partir de estas definiciones y dando un giro a la forma de almacenar el área libre de la FPGA con respecto a la de los MERs, se explica a continuación cómo se calculan las listas de vértices para una situación concreta de una FPGA:

-Habrá tantas listas de vértices como huecos tenga nuestra FPGA.

-Para cada hueco, comenzamos por convenio la lista de vértices con el vértice candidato situado más abajo y más a la derecha de nuestra FPGA.

-A partir de ese primer vértice vamos bordeando por orden todo el área libre de este hueco en concreto.

-En el caso de que en este hueco tengamos una isla, entonces uniremos los vértices que formen esa isla con el resto de los vértices de la lista a través de un vértice virtual.

-Ahora que ya tenemos una ligera idea de cómo se realiza el cálculo de la lista de vértices, podemos observar que mantener actualizadas estas listas es mucho memos costoso que los MERs, ya que los MERs se tenían que recalcular completamente cada vez que en nuestra FPGA ocurría un evento. Ahora con las listas de vértices no es necesario recalcular todos los vértices, sino sólo los nuevos vértices que se produzcan debido a la entrada o salida de una tarea en la FPGA. Para ver esto con mayor claridad, podemos observar las *figuras 6a y 6b* en las que tenemos el mismo caso que para los MERs en las *figuras 3c y 3d* y podemos observar que por el contrario que en los MERs los cuales tenían que ser recalculados, aquí con la lista de vértices pues los cambios son mucho menores.

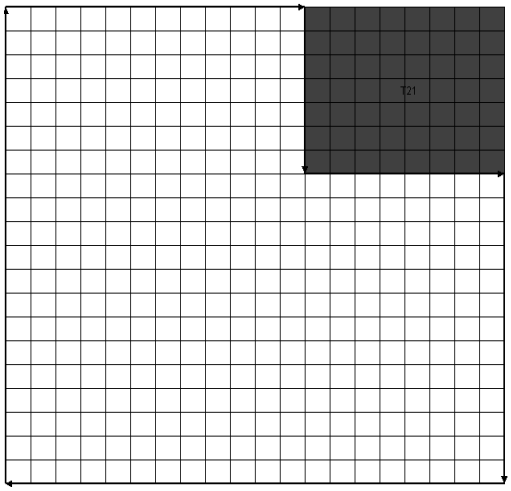


Figura 6a

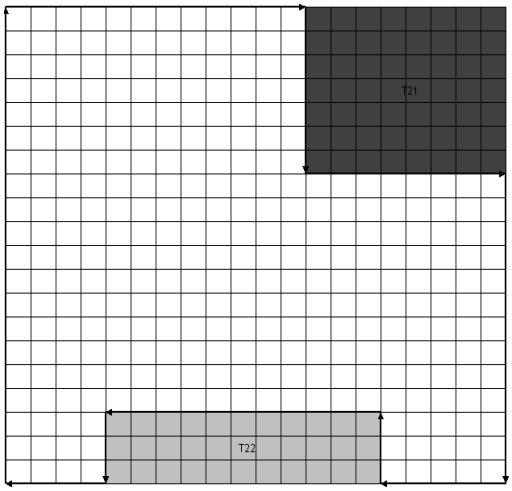


Figura 6b

-Podemos ver en la figura 6b que los cambios con respecto a la figura 6a después de incluir una nueva tarea es que aparecen cuatro nuevos vértices en la lista de vértices debido a la tarea de color azul, pero el resto de vértices se mantienen igual.

-Con este ejemplo vemos una clara ventaja de las listas de vértices frente al conjunto de MERs como estructura de almacenamiento del área libre de una FPGA y es que las listas de vértices son muchos menos costosas en complejidad de mantener que los MERs ya que los MERs tienen que recalcularse cada vez que ocurre un evento en la FPGA por el contrario que las listas de vértices ya que si ocurre un evento. Sólo habrá que recalcular los vértices implicados manteniendo intacta el resto de la estructura de la lista de vértices.

-Con la lista de vértices lo que conseguimos es rodear el área libre de nuestra FPGA de forma que cada lista de vértices tiene la propiedad de que su último vértice es el primero, por lo tanto queda una lista cerrada y todo el área que queda dentro de la lista de vértices es justo el área libre que tenemos en nuestra FPGA. Si tenemos varios huecos, pues entonces habrá tantas listas de vértices como huecos haya en la FPGA.

-Otra característica que posee la estructura de listas de vértices, son el abanico de técnicas de colocación de una nueva tarea que entra en ejecución, basadas en los vértices que forman la lista de vértices como por ejemplo la heurística 2D o la heurística 3D. Estas técnicas de ubicación basada en vértices, son más fáciles de usar en una estructura de este tipo que en las basadas en MERs como veremos más adelante.

-En la *figura 7a* podemos ver una situación de una FPGA con sus MERs y sus listas de vértices calculados y en la *figura 7b* vemos los cambios que se producen en los MERs y en la listas de vértices al incluir una nueva tarea.

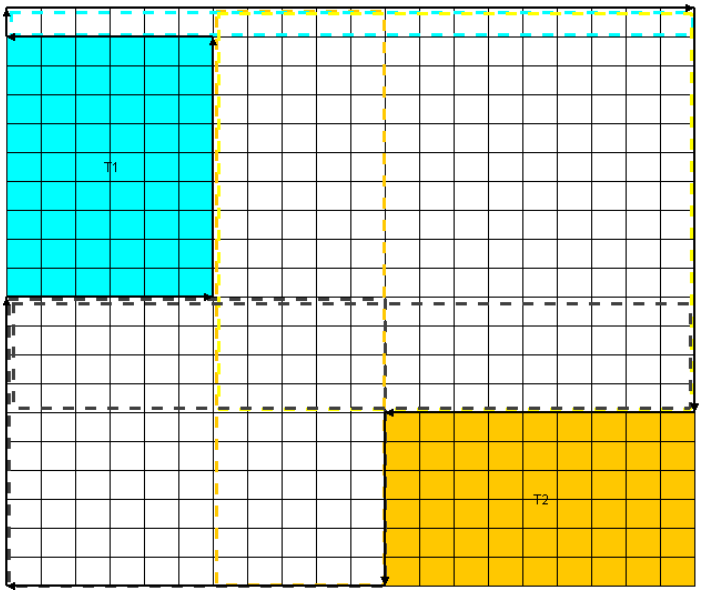


Figura 7a: MERs y lista de vértices para la situación de la FPGA.

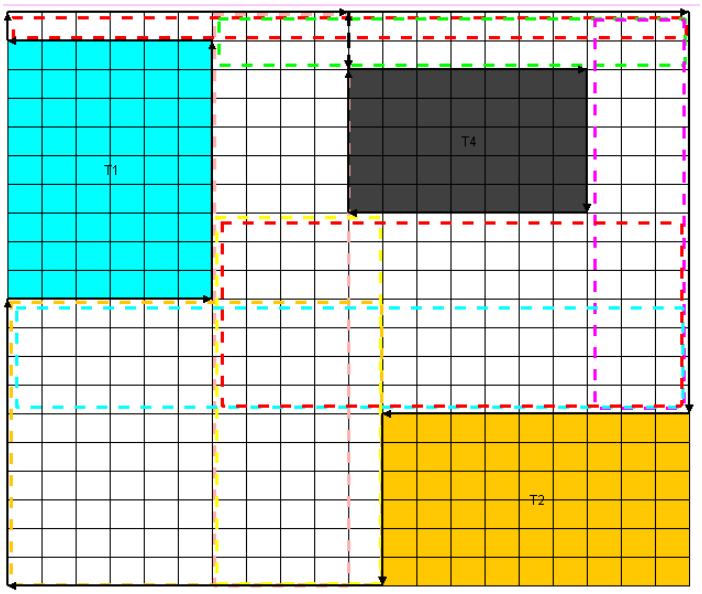


Figura 7b: MERs y lista de vértices añadiéndole a la FPGA la tarea T4.

3.- Medidas de fragmentación.

-Fragmentación significa subdivisión o separación de algo. En nuestro caso, la fragmentación tiene más que ver con lo distribuido que está el espacio libre de la FPGA, que a su vez depende de lo separadas que están unas tareas con respecto de otras cuando se están ejecutando en la FPGA.

-Cuanto más fragmentado esté el área libre de nuestra FPGA, más difícil es el aprovechar el espacio de ésta. Por lo tanto más difícil será que las tareas que van llegando a ejecución puedan entrar en la FPGA, lo que implica que el rendimiento sea mucho peor.

-Cuanto más fragmentado esté el área libre de nuestra FPGA, menos aprovechado estará esa área libre, por lo tanto más difícil será que las tareas que van llegando a ejecución puedan entrar en la FPGA, lo que implica que el rendimiento sería mucho peor.

-En la figura 8a y figura 8b se ven ejemplos de situaciones de FPGA donde, debido a la colocación de las tareas, en un caso tenemos mucha fragmentación (figura 8b) y en otro caso poca (figura 8a).
Como podemos observar en las figuras 8a y 8b, el área libre es la misma en ambos casos pero en un caso el área libre está más fragmentada que en el otro, de forma que si

quisiéramos incluir una nueva tarea de tamaño 10 X 5, habiendo la misma cantidad de área libre, como en la figura 8b el área libre está más fragmentado pues esta tarea no se podría incluir porque no entraría. Sin embargo en la figura 8a donde el área libre está muy poco fragmentado, esta tarea si que podría ubicarse en la FPGA.

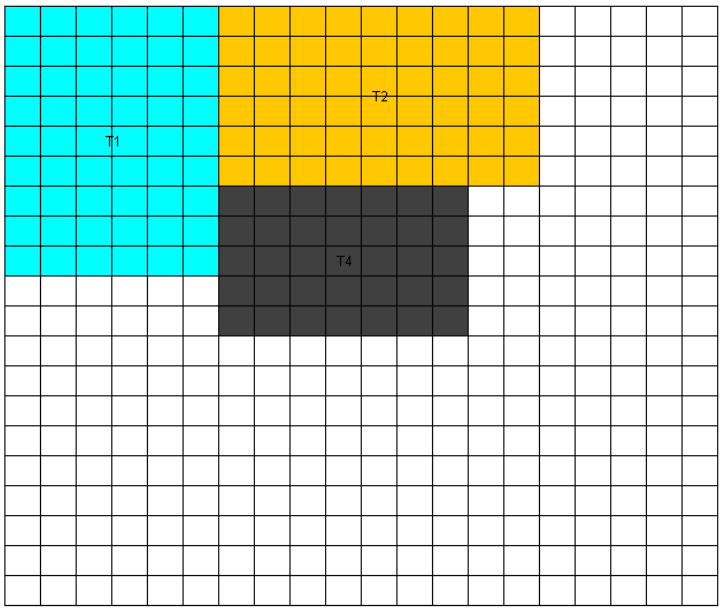


Figura 8a: situación de FPGA con tres tareas donde el área libre queda poco fragmentada.

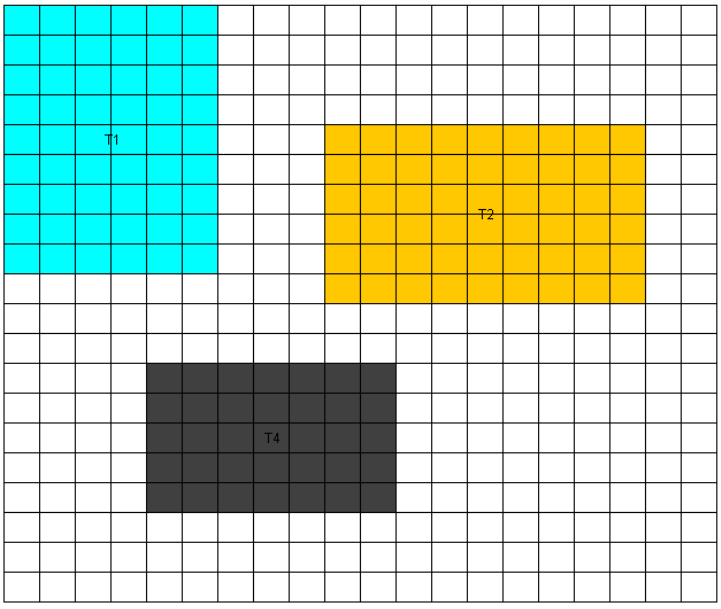


Figura 8b: situación de FPGA con tres tareas donde el área libre queda bastante más fragmentada que en la figura 8a.

-La fragmentación del área libre de una FPGA es importante e imprescindible para conseguir un buen rendimiento de nuestra FPGA, por lo tanto es interesante medir en cada momento cuánto de fragmentada está nuestra FPGA. Entonces ahora se nos plantea el problema de medir la fragmentación de una FPGA por lo que aparecen varias medidas de fragmentación de las cuales vamos a explicar dos.

-Primera medida de fragmentación, fragmentación basada en huecos:

Podemos tener una estimación acertada de la fragmentación de la FPGA en un estado concreto usando esta medida de fragmentación, donde el nivel de fragmentación del área libre para un estado dado de una FPGA viene estimado como sigue:

$\boldsymbol{F} = 1 - \Pi_i \ [(4/V_i)^n * (A_i/A_{F\_FPGA})]$

Donde el término entre corchetes representa un grado de adecuación para un hueco H_i dado, con área A_i y vértices V_i:

- $(4/V_i)^n$ representa la adecuación de la forma del hueco i para acomodar una tarea rectangular. Notar que un hueco con exactamente cuatro vértices, tiene la mejor adecuación. En este trabajo se ha usado el valor de n a 1, pero se podría usar un valor mayor o menor si se quiere penalizar más o menos la aparición de huecos con formas complejas y por lo tanto su uso más difícil para alojar una tarea rectangular.

- (A_i/A_{F_FPGA}) representa la relación de área del hueco i (A_i) con respecto el área total. A_{F_FPGA} es el área libre total de la FPGA. Que resulta $A_{F\_FPGA} = \sum A_i$. Es decir, la suma del área de cada hueco.

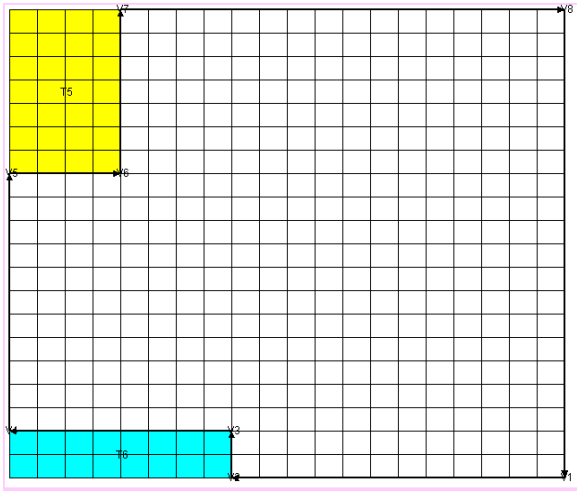


Figura 9a: sólo hay un hueco formado por 8 vértices, F = 0.5

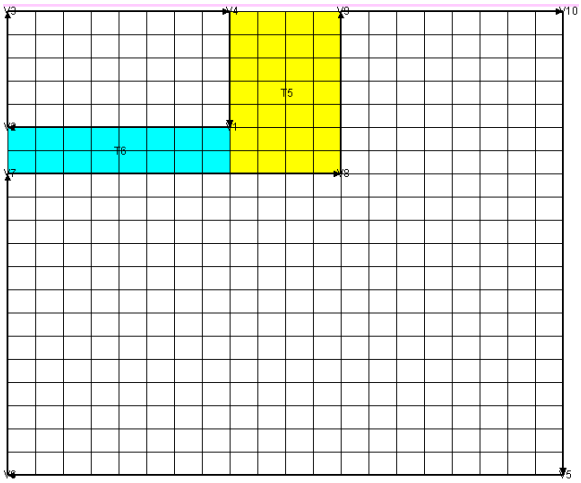


Figura 9b: ahora hay dos huecos de 4 y 6 vértices cada uno, F = 0.93

Como se ha podido observar, una FPGA es globalmente menos adecuada cuantos más huecos separados haya de área libre. Notar también que el valor de la fragmentación (F) se dispara con el número de huecos por lo que esta medida de fragmentación, si un estado concreto de la FPGA tiene mas de uno o dos huecos, su valor se eleva muchísimo. En las *figuras 9a y 9b*, podemos ver que para el mismo conjunto de tareas ejecutándose en la FPGA, el valor de la fragmentación F es distinto si este conjunto de tareas forman o no un hueco.
Lo más difícil de esta medida de fragmentación es saber cómo cuantificar en la fórmula ambos parámetros (huecos y vértices) ya que tampoco está muy claro qué es peor, si sólo hueco con muchos vértices y dos huecos con el mínimo número de vértices (cuatro).

Por estas razones, se plantea a continuación otra medida de fragmentación para un estado concreto de una FPGA.

-Segunda medida de fragmentación, fragmentación basada en perímetros:

-Ahora se presenta una nueva medida de fragmentación que mide la adecuación del área libre para recibir una nueva tarea con un sólo parámetro que es cuánto de cuadrado es cada uno de los huecos. Se considera que el mejor área libre está contenida hueco de forma cuadrada. De este modo usamos el perímetro de cada hueco para calcular cómo sería el **área libre ideal** para ese perímetro si el hueco fuera un cuadrado perfecto.

-El área ideal es calculado de la siguiente forma:

$$área\_libre\_ideal = ((\sum perímetro\_hueco_i)/4)^2$$

-Se divide la suma del perímetro de todos los huecos por cuatro. De esta

forma se obtiene el lado del cuadrado ideal, y elevado al cuadrado es el área ideal del hueco.

-Finalmente la medida de fragmentación F es la siguiente:

$$F = 1 - (área\_libre\_real/área\_libre\_ideal)$$

-Donde el término área_libre_real representa el área libre disponible en la FPGA, y el término área_libre_ideal es el calculado anteriormente.

-En las figuras 10a y 10b se pueden ver distintos valores de la fragmentación con esta medida para las mismas situaciones que en las figuras 9a y 9b. Ahora podemos observar que el valor de la fragmentación no se penaliza tanto si hay uno o dos huecos.

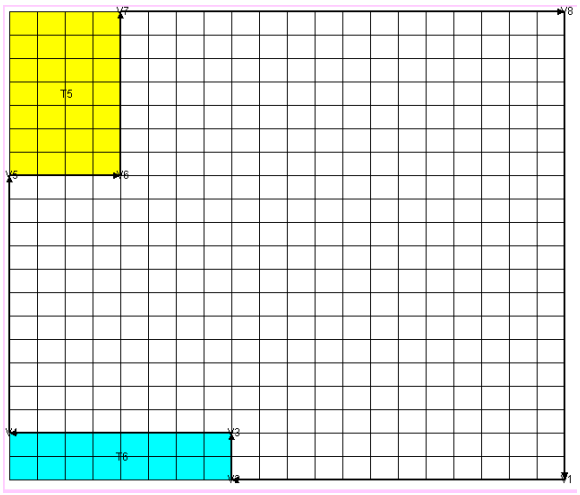


Figura 10a: sólo hay un hueco formado por 8 vértices, F = 0.1

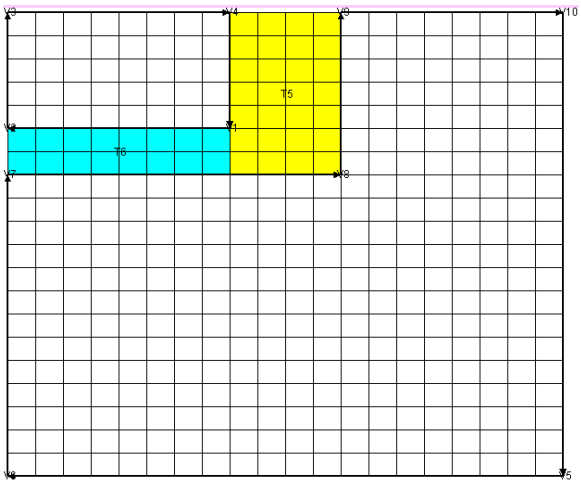


Figura 10b: ahora hay dos huecos de 4 y 6 vértices cada uno, F = 0.47

4.- Técnicas de ubicación.

-Al llegar una nueva tarea a ejecución en la FPGA, convine tener técnicas de ubicación para colocar esa nueva tarea en el mejor lugar posible de forma que la FPGA quede lo menos fragmentada posible o visto de otra forma, que la tarea quede lo mejor ubicada posible, para permitir la colocación de futuras tareas.

-Notar la importancia de colocar una tarea con algún tipo de criterio, ya que si se coloca en el primer espacio libre, sin tener en cuenta el estado del área libre de la FPGA, probablemente se obtenga mucho peor rendimiento que si las tareas se colocan según algún patrón o heurística de colocación concreta.

-Estas técnicas de ubicación de una nueva tarea, se pueden subdividir en dos clases:

-Técnicas de ubicación de tareas **basadas en los MERs.**

-Técnicas de ubicación de tareas **basadas en los vértices candidatos de las listas** de vértices.

-A su vez, estas técnicas de ubicación se pueden subdividir en otras dos clases:

-Técnicas basadas en **adyacencia**.

-Técnicas basadas en medidas de **fragmentación.**

-Para cada clase de técnicas de ubicación se explicarán algunas técnicas y se mostrará su resultado a través de un ejemplo concreto para entender mejor el concepto de cada técnica de ubicación.

4.1- Técnicas de ubicación de tareas basadas en los MERs

-A partir del conjunto de MERs como estructura para la gestión del área libre de la FPGA, podemos tener distintas técnicas de ubicación de tareas, pero como podremos ver más adelante, estas técnicas dan peor resultado que las técnicas basadas en los vértices de las listas de vértices.

-A continuación alguna de las posibles técnicas de colocación de tareas basadas en MERs:

-"Best fit" o **"mejor ajuste":** consiste en seleccionar del conjunto de MERs, aquellos en los cuales la tarea a colocar quepa, y de todos ellos, coger el menor. Así conseguiremos ajustar la tarea a un MER desperdiciando la menor cantidad de área posible de cada MER. En la *figura 4a* podemos ver un ejemplo en el cual, podemos observar dónde iría colocada una tarea que va a ejecutarse de tamaño 3 X 3 utilizando la técnica "best fit" o "mejor ajuste".

-"Worst fit" o **"peor ajuste":** es exactamente igual que la anterior con la salvedad de que dentro del conjunto de MERs donde cabe la tarea a ejecutar, esta vez seleccionamos el MER en el cual la tarea se ajusta peor, es decir, el MER más grande. En la *figura 4b* podemos ver un ejemplo en el cual, para una situación similar a la de la figura X, dónde iría colocada una tarea que va a ejecutarse de tamaño 3 X 3 utilizando la técnica "worst fit" o "peor ajuste".

-En ambas técnicas de ubicación, tanto en peor ajuste como en mejor ajuste, el criterio de colocar la tarea en el MER es el de colocarla lo más abajo y más a la izquierda posible, de forma que el vértice inferior izquierdo de la nueva tarea quede encajado en el vértice inferior izquierdo del MER seleccionado para contener la nueva tarea. Se ha escogido este criterio por hacerlo siempre de la misma forma ya que a priori nunca se puede saber en qué vértice será mejor colocar la tarea dentro del MER.

-En las figuras 11a y 11b podemos ver dónde iría colocada una tarea de tamaño 3 X 3 a partir de la técnica de mejor ajuste y de peor ajuste basada en MERs respectivamente.

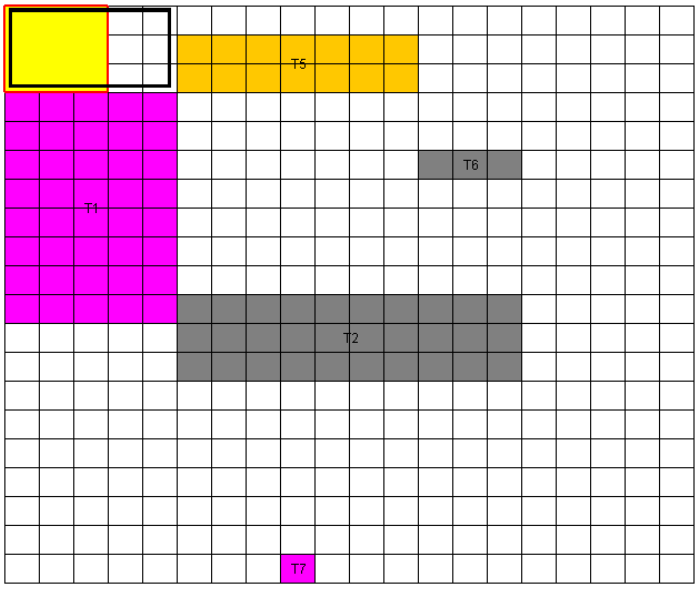


Figura 11a: la tarea amarilla es la tarea a incluir y el recuadro negro indica el MER seleccionado a partir del mejor ajuste.

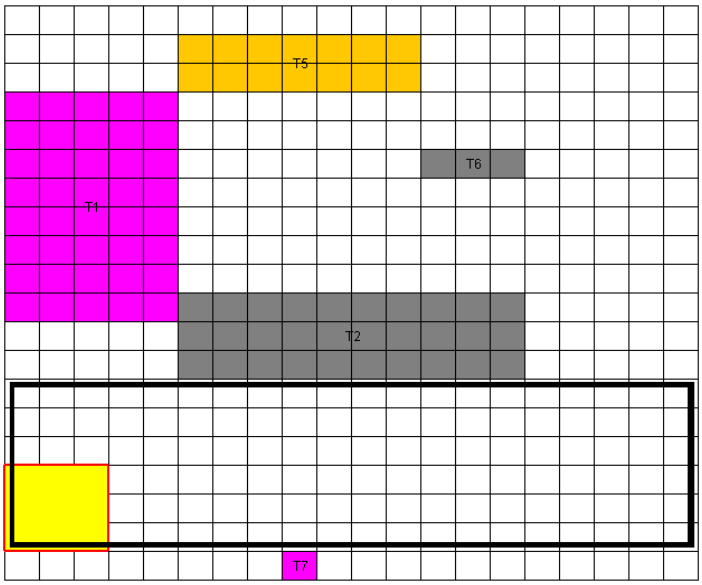


Figura 11b: la tarea amarilla es la tarea a incluir y el recuadro negro indica el MER seleccionado a partir del peor ajuste.

4.2 Técnicas de ubicación de tareas basadas en las listas de vértices.

-A partir de la estructura de listas de vértices como almacenamiento del área libre de la FPGA, tenemos varias técnicas de ubicación de una nueva tarea cada con unos criterios particulares. Todas estas técnicas tienen en común que una tarea nueva que va a ejecutarse en la FPGA, siempre se va a ubicar en un vértice candidato de todos los vértices que forman las listas de vértices lógicamente siempre y cuando la nueva tarea a ubicar, tenga espacio suficiente en ese vértice en concreto para ser colocada.

-Partiendo de esta base de que toda tarea nueva que entre será colocada en un vértice candidato, lo que distingue una técnica de otra es el vértice en concreto que seleccionaremos para ubicar la nueva tarea de entre todos lo vértices candidatos que tengamos, siempre y cuando la tarea tenga espacio libre suficiente como para que pueda ser colocada en un vértice concreto.

-Es aquí cuando, en función de la forma de seleccionar un vértice candidato para ubicar en él la nueva tarea, cuando distinguimos dos subclases de técnicas que son las siguientes:

4.2.1.- Técnicas de ubicación a partir de las medidas de fragmentación.

-Como hemos visto anteriormente, hemos explicado dos posibles medidas de la fragmentación de la FPGA para un estado en concreto. Una forma de seleccionar un vértice candidato para ubicar la nueva tarea es la siguiente:

-Para cada vértice candidato en el cual la nueva tarea pueda ser ubicada en él, colocamos la tarea en ese vértice y medimos la fragmentación resultante de colocar ahí la tarea con una de las medidas de fragmentación explicadas anteriormente y guardamos ese valor de medida de la fragmentación con la tarea ya colocada.

-Ahora simplemente nos quedamos con la situación en la cual el valor de la fragmentación nos ha dado el menor valor de forma que al final tenemos la tarea colocada en el vértice candidato tal que la fragmentación resultante en la FPGA sea la menor posible.

-De esta forma obtenemos dos técnicas de ubicación de una nueva tarea en la FPGA. Ambas siguen el mismo algoritmo explicado justo antes con la salvedad de que cada una mide la fragmentación con una medida u otra.

-En las figuras 12a, 12b y 12c podemos observar dónde iría colocada una tarea para una misma situación de la FPGA. Usando la medida de fragmentación basada en huecos o en el perímetro respectivamente, con los valores en la tabla respectivos de la fragmentación y con un tamaño de tarea 5 X 5.

V	H1
V1	0.75
V2	-
V3	-
V6	-
V7	-
V9	0.95
V10	0.66
V11	0.66
V14	0.71

V	H2
V1	0.70
V2	-
V3	-
V6	-
V7	-
V9	0.70
V10	0.65
V11	0.65
V14	0.70

Figura 12a y 12b: listado de todos los vértices candidatos y su valor de medida de fragmentación si la nueva tarea es colocada en ese vértice. H1 es el valor correspondiente a la medida basada en huecos y H2 la medida basada en perímetro. A la izquierda el valor seleccionado para H1 y a la derecha el seleccionado para H2.

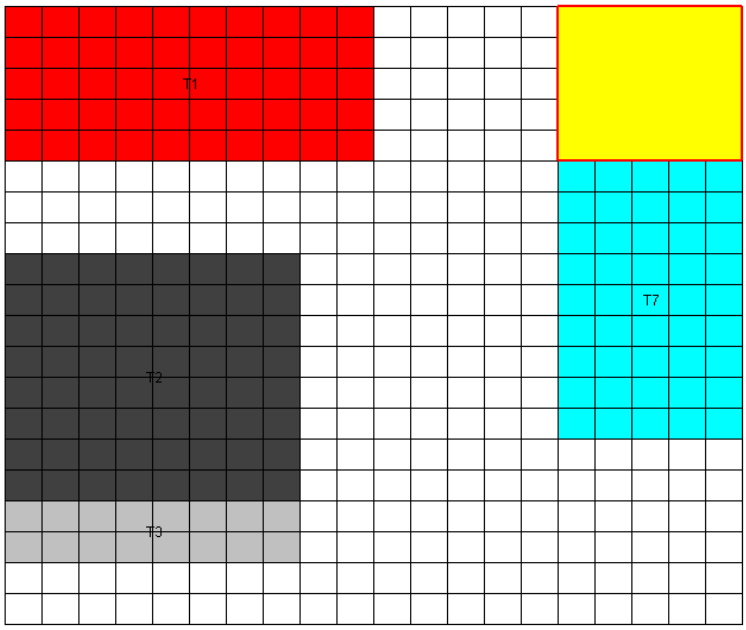


Figura 12c: la tarea en amarillo es la tarea a ubicar a partir de la medida de fragmentación basada en huecos y basada en el perímetro. La tarea es colocada en el vértice 9.

-En este ejemplo en concreto podemos observar que hay valores de la fragmentación que nos dan iguales, pero de nuevo se usa el criterio de que en caso de dar el mismo valor, nos quedamos con el primer vértice que nos encontremos. Notar que en este ejemplo en concreto, ambas medidas de fragmentación nos colocan la tarea en el mismo vértice candidato.

4.2.2.- Técnicas de ubicación a partir de heurísticas de adyacencia.

-De igual forma que en el caso anterior, tenemos una tarea a ubicar y un estado en concreto de nuestra FPGA y aplicando un algoritmo muy similar al anterior, seleccionamos un vértice candidato u otro para ubicar la nueva tarea en función del mejor valor que nos de la heurística en concreto que utilicemos.

-Estas técnicas basadas en heurística son dos, las cuales una es una versión mejorada de la anterior:

-Heurística de adyacencia 2D.
-Heurística de adyacencia 3D.

4.2.2.1.- Heurística de adyacencia 2D.

-Antes de explicar en qué se basa la heurística de adyacencia 2D para medir cada vértice candidato de nuestras listas de vértices, vamos a explicar un concepto.

-Se entiende que una tarea está mejor ubicada en un lugar si la tarea "encaja" mejor en ese lugar que en cualquier otro. Entendiendo por lugar, un vértice candidato u otro. Que una tarea "encaje" mejor, quiere decir que las paredes de esa tarea toquen lo más posible con paredes de otras tareas o con las propias paredes de la FPGA.

-A partir de este concepto de "encaje", lo que mide la heurística de adyacencia 2D es, a partir de una tarea nueva que va a entrar a ejecución, en estado concreto de la FPGA y un vértice candidato en concreto en el cual la tarea pudiera ser ubicada, lo que mide la heurística de adyacencia 2D es cuánto toca las paredes de la nueva tarea con otras paredes de tareas ubicadas ya en la FPGA o con las propias paredes de la FPGA, de forma que cuanto más perímetro de la nueva tarea toque con el perímetro de otras tareas de la FPGA o con la propia FPGA, mejor será el valor heurístico obtenido.

-En la figuras 13a y 13b podemos observar dónde se colocaría una nueva tarea de tamaño 5 X 3 para un estado concreto de la FPGA.

V1	8
V2	-
V3	-
V5	5
V7	8
V8	8
V9	8
V12	8
V13	13
V14	13
V15	13
V16	13

Figura 13a.

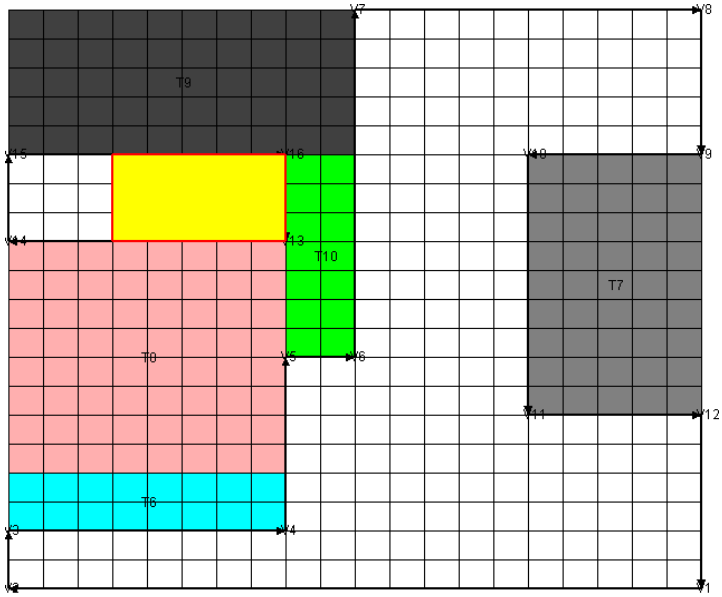


Figura 13b: la tarea amarilla es la nueva tarea a colocar según la heurística de adyacencia 2D para este estado concreto de la FPGA.

-Vemos en la figura 13b que los vértices candidatos en donde la nueva tarea de tamaño 5 X 3 iría colocada serían el vértice 13, 14, 15, y 16 ya que si tomamos como unidad el ancho o alto de una celda de la FPGA, la cantidad de perímetro que toca con el perímetro de otras tareas o de la propia FPGA es para estos cuatro vértices de 13 y si nos fijamos en la

figura 13a, no hay más de trece unidades de perímetro que toquen con otras tareas o con la FPGA.

4.2.2.2.- Heurística de adyacencia 3D.

-Esta heurística se basa en la heurística de adyacencia 2D pero añadiendo diversas modificaciones. Para explicar la heurística de adyacencia 3D [4] primero hay que dar unos conceptos sobre arista y el valor que toma cada arista necesarios para al cálculo de la heurística de adyacencia 3D para un vértice candidato en concreto.

-Se entiende por **arista** a la línea formada por dos vértices de la lista de vértices.

-El tiempo de permanencia de cada arista perteneciente a la lista de vértices se calcula de la siguiente manera:

> -(1) Si la arista está sobre alguna pared de la FPGA, esa arista tiene valor infinito.
> -(2) Si la arista no está sobre alguna pared de la FPGA, entonces por construcción de la lista de vértices, esa arista estará sobre el perímetro de una o varias tareas.
>
> > -(2.1) Si la arista está sobre el perímetro de una sola tarea, el valor de esa arista será el tiempo de ejecución que le quede a esa tarea en concreto.
> >
> > -(2.2) Si la arista está sobre el perímetro de varias tareas, entonces el valor de esa arista será el del tiempo de la tarea que menos tiempo de ejecución tenga.

-En la figura 14 podemos ver al valor de las aristas para los tres posibles casos para tener así una idea más clara de cómo calcular el valor de cada arista: recuadrado en marrón tenemos un ejemplo del caso 1, recuadrado en rosa un ejemplo del caso 2.1 y recuadrado en rojo un ejemplo del caso 2.2 en donde la arista está sobre el perímetro de dos tareas, una de tiempo de ejecución 20 y la otra de 18, por lo tanto el valor de la arista es de 18.

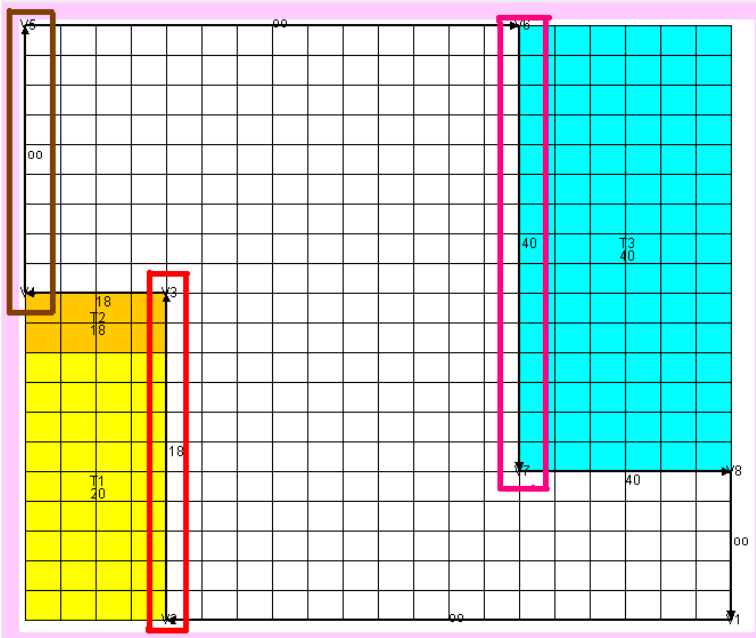


Figura 14.

-De esta forma conseguimos que el valor de la arista sea el tiempo que esa arista va a existir ya que:

> -Si la arista está sobre una pared de la FPGA, entonces esa arista tiene valor infinito porque a la pared de la FPGA nunca desaparecerá.
>
> -Si la arista está sobre el perímetro de una o varias tareas, pues el valor de esa arista será el tiempo que permanecerá esa arista que es el menor tiempo de ejecución restante de esas tareas.

-Notar que ahora el que una arista a priori vaya a permanecer más tiempo, vamos a tenerlo en cuenta a la hora de calcular el valor de la función de coste con un tiempo de permanencia mayor.

-Explicado todo esto, el cálculo de la heurística de adyacencia 3D a partir de una tarea a ubicar, de un vértice candidato en concreto en el cual la tarea pudiera ser colocada y de un estado en concreto de la FPGA, el valor de esta heurística viene dado de la siguiente forma:

> -El tiempo de permanencia comienza valiendo 0.
>
> -Para cada unidad de perímetro de la tarea a colocar suponiendo que se coloca en un vértice candidato concreto.
>
> > -Si esa unidad de perímetro toca con alguna de las aristas, entonces se le suma al valor de la heurística, el menor valor entre el tiempo de permanencia de esa arista y el tiempo de ejecución de la nueva tarea.
> > -Si toca con una pared de la FPGA pues se le suma al valor de la heurística, el tiempo de ejecución de la nueva tarea.

-Si no toca con alguna arista pues no se suma nada.

-De esta forma tenemos que la tarea no sólo va a colocarse donde más adyacencia encuentre con otras tareas o con la propia FPGA sino que se colocará además donde más tiempo a priori vaya a estar bien colocada, por lo que en la mayoría de los casos, la heurística de adyacencia 3D nos dará mejor resultado que la heurística de adyacencia 2D.

V1	128
V2	-
V3	-
V5	54
V7	128
V8	128
V9	88
V12	88
V13	165
V14	168
V15	168
V16	165

Figura 16a.

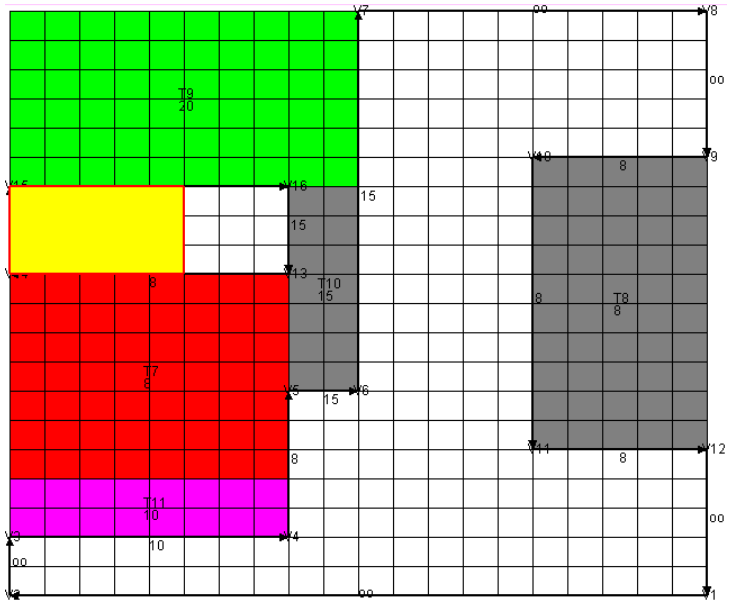


Figura 16b: la tarea amarilla es la nueva tarea a colocar según la heurística de adyacencia 3D para este estado concreto de la FPGA.

-En las figuras 16a y 16b se ve cómo para la misma situación que en el ejemplo de la heurística de adyacencia 2D, ahora para una tarea nueva del mismo tamaño y tiempo de ejecución 16, la heurística de adyacencia 3D coloca la tarea pegada a la pared de la FPGA ya que en este caso, como la nueva tarea tiene su tiempo de ejecución a 16, pues pegado a la pared de la FPGA va a estar más tiempo "encajada" que en la pared de la tarea T10 de tiempo 15. En la figura 16a se puede ver el valor de la heurística de adyacencia 3D para cada vértice candidato.

Ejemplos de gráficas de datos a partir del resultado de una serie de ejecuciones de lotes de tareas.

-Al igual que medir el rendimiento de un computador es complicado y para ello se utilizan una serie de programas de medida del rendimiento llamados benchmarks, aquí nos pasa exactamente igual ya que la única forma de medir la calidad de una técnica de ubicación de tareas es a partir de simulaciones de distintos lotes de tareas y comprobar su resultado: cuántas tareas se han podido ejecutar y cuántas no, qué porcentaje de área en promedio se ha usado durante la ejecución o el volumen de tareas ejecutadas son medidas que nos servirán para probar la calidad de las distintas técnicas de ubicación de tareas.

-Para ello mostraremos una serie de gráficos en los que para cada técnica de ubicación y para cada lote distinto de tareas a ejecutar, podamos ver el valor de los parámetros citados anteriormente y así poder sacar algunas conclusiones. Más en concreto tendremos en la figura 17a el porcentaje de volumen no ejecutado y en la figura 17b el promedio de área utilizado durante la ejecución, partiendo de la base que ejecutamos los lotes de tareas sobre una matriz de tamaño 100 X 100.

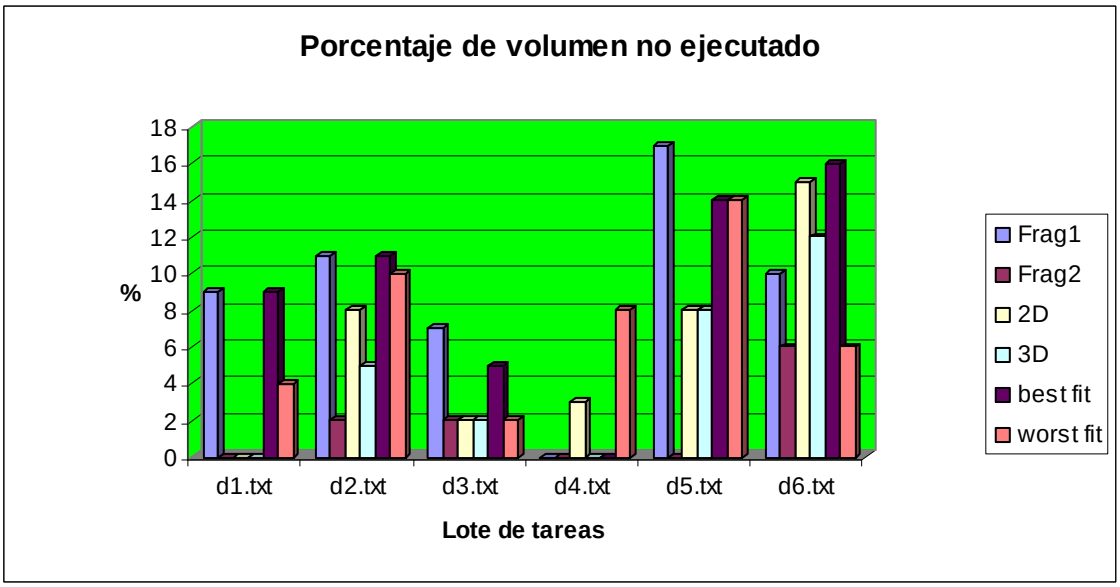


Figura 17a.

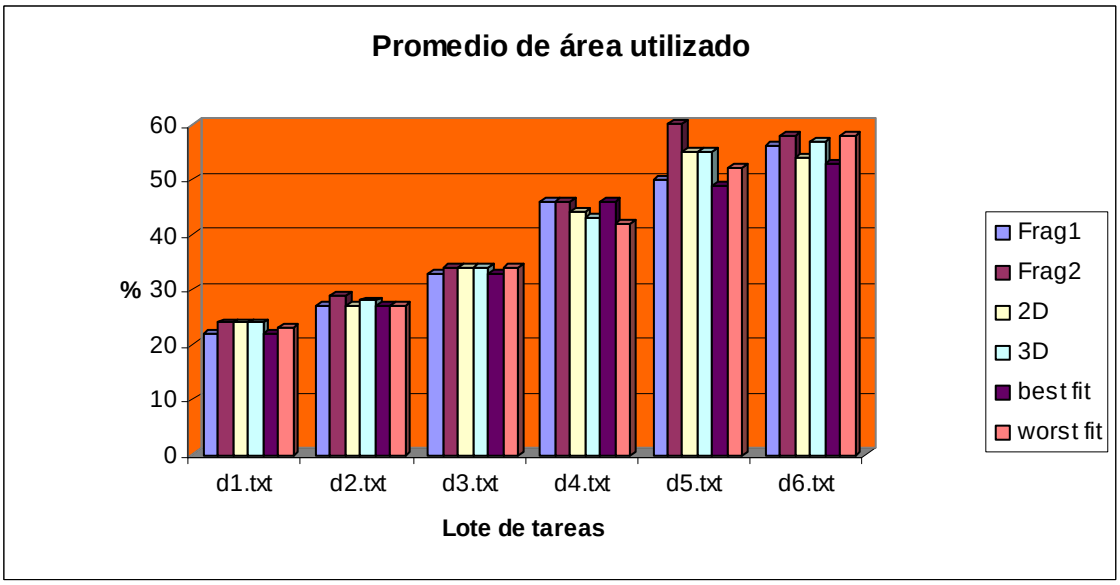


Figura 17b.

5.- Conclusiones

-Con respecto a las gráficas obtenidas en el apartado anterior, son varias las conclusiones se pueden tomar a partir de la ejecución de seis archivos distintos cada uno formado por un lote de tareas y probados para cada una de las técnicas de ubicación explicadas.

-Si nos preguntamos sobre cuál es la clase de técnicas de ubicación que mejor rendimiento nos da, mirando ambas gráficas se puede observar claramente que las técnicas de ubicación basadas en listas de vértices nos dan mejor rendimiento que las basadas en MERs, tanto en el porcentaje de volumen de tareas no ejecutado (que en el caso de las técnicas de ubicación basadas en listas de vértices es menor) como en el área utilizada en promedio durante la ejecución (que en el caso de las técnicas de ubicación basadas en listas de vértices es mayor).

-Dentro de las técnicas de ubicación basadas en MERs como estructura de almacenamiento y gestión del área libre, a priori se puede pensar que la técnica de “best fit” puede tener al menos una ligera mejora del rendimiento con respecto a la técnica “worst fit”, pero si nos fijamos en los resultados para los distintos lotes de tareas, se puede observar que no es así. Para algunos lotes de tareas funciona mejor la técnica del “best fit” y para otros, la que nos da un mejor rendimiento es la técnica “worst fit”. Con esto se puede llegar a la conclusión de que utilizando distintas técnicas de ubicación basadas en MERs, con ninguna se llega a una situación en la que podamos decir que una técnica ofrece mejor rendimiento que otra en general. Esto es debido a que no por ajustar una tarea lo máximo posible en un MER, no por ello vamos a obtener un mejor resultado que si esa tarea la ajustamos a un MER cualquiera siempre y cuando sea posible.

-Lo importante es usar heurísticas que tiendan a disminuir la fragmentación del área libre. La técnica de lista de vértices además acelera el proceso debido a la menor complejidad que conlleva el uso de listas de vértices con respecto a los MERs, ya que la colocación de una tarea viene ya determinada por la posición del vértice candidato escogido.

-Por lo tanto, se puede llegar a la conclusión de que aplicar técnicas de ubicación de tareas sobre una estructura de almacenamiento del área libre como son los MERs, no nos da mucho rendimiento con respecto a as técnicas basadas en listas de vértices. Además es difícil a priori saber si una técnica de ubicación de tareas basada en MERs va a ser mejor (en cuestiones de rendimiento) u otra.

Apéndices

Apéndice 1: Manual de usuario de la herramienta.

-FPGATool es una herramienta que posee distintas funciones. Todas pueden realizarse de forma sencilla y aquí las explicaremos de forma detallada.

-Las distintas funciones que tiene FPGATool son las siguientes:

1.-Simular la gestión de área en una FPGA de las dimensiones que se deseen.

2.-Usar el interfaz de tareas con el que podemos crear, modificar y eliminar tareas de cualquier tamaño y añadirlas en la FPGA.

3.-Dar las dimensiones y el tiempo de una nueva tarea y ver según la técnica de ubicación elegida, dónde iría esa tarea colocada para luego tener la opción de colocarla o desecharla.

4.-Salvar y cargar cualquier configuración realizada con nuestra herramienta para poder continuar en otro momento.

5.-Leer un lote da tareas y simular su ejecución evento a evento o la ejecución total según guste y obtener al final una ventana con información sobre la ejecución.

-Antes de explicar cómo usar cualquier función de la herramienta, a continuación mostramos una imagen de la apariencia de la interfaz de usuario de la herramienta:

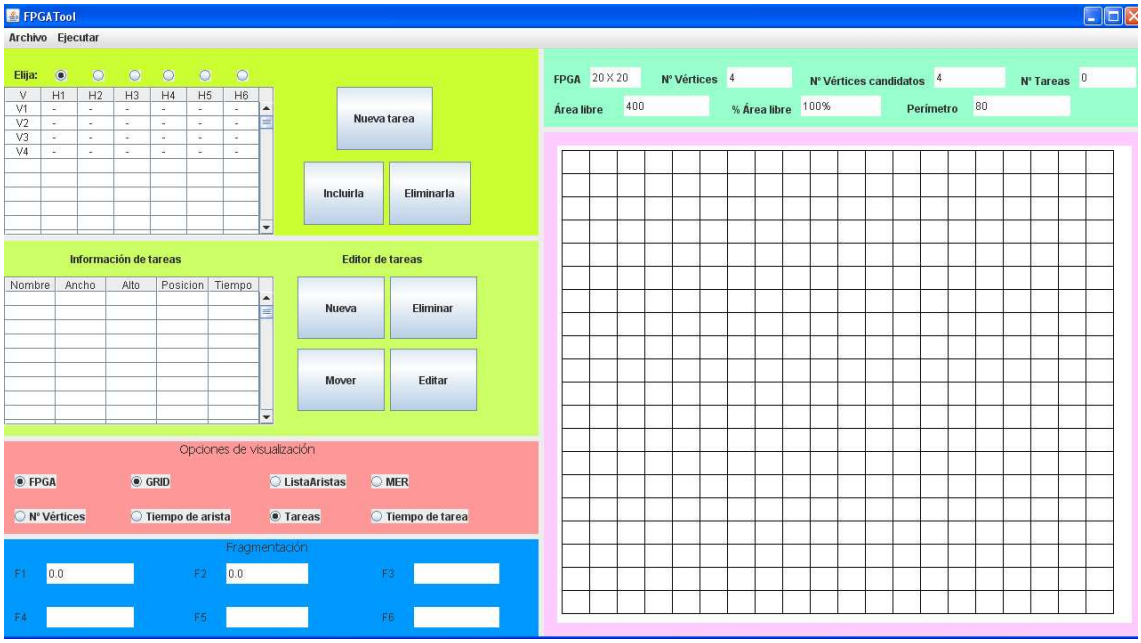


Figura 18.

1.- Simular la gestión de área de una FPGA con las dimensiones que se desee.

-Para simular una FPGA con el tamaño que deseamos, basta con pinchar en Archivo y dentro del submenú de “Archivo”, pinchar en “Nuevo” tal y como muestra la siguiente imagen:

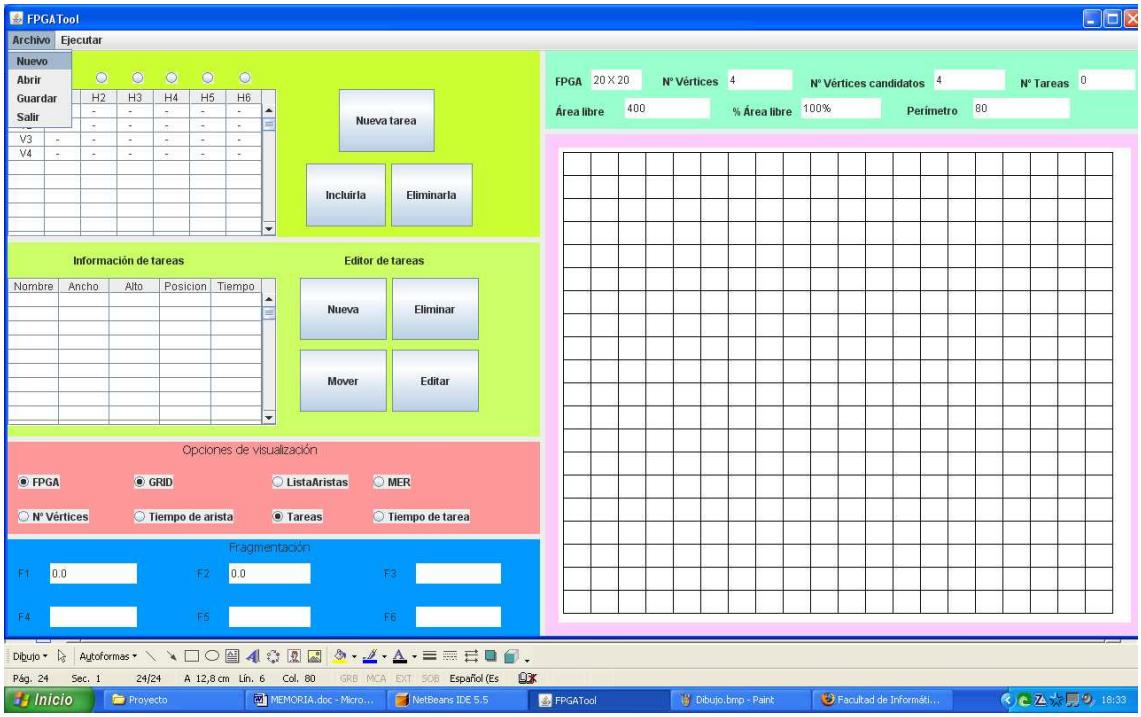


Figura 19.

-Entonces nos saldrá una ventana en la que indicaremos el número de celdas de ancho y de alto que tendrá nuestra FPGA. Tras introducir los datos, le damos al botón aceptar si es que todo está correcto y queremos seguir adelante con la nueva FPGA o podemos darle al botón cancelar si nos arrepentimos de la acción. Al ejecutar la aplicación, por defecto viene una matriz de tamaño 20 X 20. Las siguientes imágenes muestran la ventana para crear la nueva FPGA.

-Notar que si creamos una nueva FPGA, se borrará el estado actual de ésta y se creará una FPGA con las dimensiones que le hemos dado pero totalmente vacía. Si no se quieren perder los datos actuales, basta con guardar el estado de la FPGA.

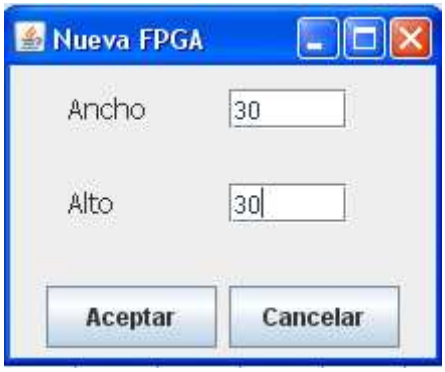


Figura 20.

2.- El interfaz de tareas

-Con el interfaz de tareas podemos realizar cualquier manipulación de tareas tales como crear, mover, editar o eliminarlas.

-El interfaz de tareas se muestra a continuación:

Figura 21.

-Si queremos crear una nueva tarea, le damos al botón “Nueva” y nos pedirá el tiempo de ejecución de esa tarea. Tras introducir el tiempo de ejecución y darle al botón de aceptar, estas en disposición de mover el cursor hacia el tablero en el cual veremos que ahora nos sale una tarea por defecto de tamaño 1 X 1 por donde movamos el cursor. Entonces si pinchamos con el ratón sobre cualquier celda libre (de color blanco) y sin soltar el botón del ratón, arrastramos el cursor hacia cualquier dirección, podremos colocar una tarea del tamaño que se quiera y donde se quiera soltando el botón del ratón, siempre y cuando no se salga de la FPGA ni solape con otra tarea que ya exista en la FPGA. Si esto ocurre, la operación queda automáticamente cancelada y tendremos de nuevo la posibilidad de colocar la nueva tarea.

-Si lo que queremos es modificar la forma de una tarea que ya está en nuestra FPGA, entonces basta con darle al botón “Editar” y a continuación pincharemos con el ratón en la tarea que deseemos modificar y sin soltar el botón del ratón, le daremos la nueva forma que queramos. Cuando tengamos la nueva forma, basta con soltar el botón del ratón.

-Para mover una tarea de sitio sin modificar su forma, le damos al botón “Mover” y a continuación operamos de la misma forma que la explicada para modificar una tarea.

-O si lo que queremos es eliminar una tarea ya existente en la FPGA, basta con darle al botón “Eliminar” y seguidamente pinchar con el ratón sobre la tarea que queramos borrar y ésta desaparecerá de la FPGA.

-De esta forma, con el editor de tareas podemos crear y manipular las tareas sin ninguna dificultad, permitiendo crear cualquier estado para cualquier tamaño de la FPGA deseado.

3.-Crear una nueva tarea y dejar que las técnicas de ubicación decidan su colocación.

-Además de crear una tarea e insertarla donde nosotros queramos, también tenemos la opción de crear una nueva tarea dándole su tamaño y sus dimensiones y su ejecución y esperar que la aplicación la coloque en la FPGA según la técnica de ubicación elegida.

-La elección de la técnica de ubicación se elige de la siguiente haciendo clic sobre ella con el ratón. La siguiente imagen muestra las distintas técnicas de ubicación que por defecto siempre vendrá seleccionada la primera:

Figura 22.

-El significado de cada abreviatura es el siguiente:

-H1: técnica de ubicación basada en selección de vértice a partir de la medida de fragmentación basada en huecos.

-H2: técnica de ubicación basada en selección de vértice a partir de la medida de fragmentación basada en perímetros.

-H3: técnica de ubicación basada en selección de vértice a partir de la heuri´stica de adyacencia 2D.

-H4: técnica de ubicación basada en selección de vértice a partir de la heuri´stica de adyacencia 3D.

-H5: técnica de ubicación basada en MERs a partir de la técnica de “best fit” o “mejor ajuste”.
-H6: técnica de ubicación basada en MERs a partir de la técnica de “worst fit” o “peor ajuste”.

Figura 23.

-Ahora si le damos al botón que dice “Nueva tarea” nos saldrá una ventana para introducir el ancho, alto y el tiempo de ejecución de la nueva tarea como muestra la siguiente figura:

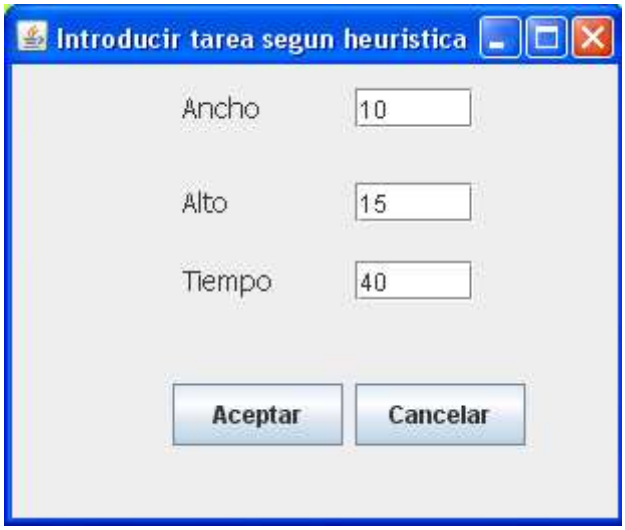


Figura 24.

-Como siempre, una vez introducidos los datos si le damos a cancelar de cerrará esta ventana y se abortará la operación de nueva tarea. Pero si le damos a captar, entonces nos aparecerá en el panel de la FPGA la nueva tarea de color amarillo con borde rojo colocada donde correspondiera según la técnica de selección escogida. Ahora estamos en condiciones de poder hacer tres cosas:

-Cambiar la técnica de ubicación de la nueva tarea y visualizar de inmediato en que lugar del área libre de la FPGA será colocada.

-Incluir esta nueva tarea al estado de la FPGA de forma definitiva dándole al botón “Incluirla”.

-O eliminar esa tarea dándole al botón “Eliminarla”.

La siguiente figura muestra dónde sería colocada una nueva tarea para un estado concreto de la FPGA y para una técnica de ubicación en concreto seleccionada:

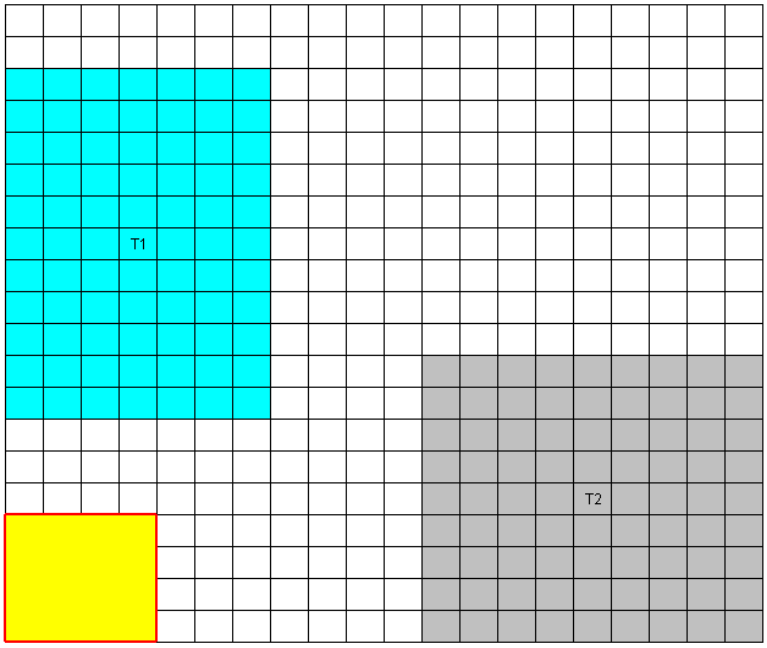


Figura 25.

4.-Salvar y cargar una configuración.

-Podemos estar en un caso en el que tenemos una configuración de la FPGA que nos interesaría almacenar para poder usarla posteriormente. Con FPGATool es posible ya que cualquier configuración de la FPGA se puede almacenar de forma fácil: pinchamos en el menú “Archivo” y seguidamente pinchamos en el submenú “Guardar” tal y como nos muestra la siguiente figura:

Figura 26.

-Entonces nos saldrá un menú con el que podremos dar el nombre de nuestro fichero y elegir en qué directoria almacenarlo como se muestra a continuación:

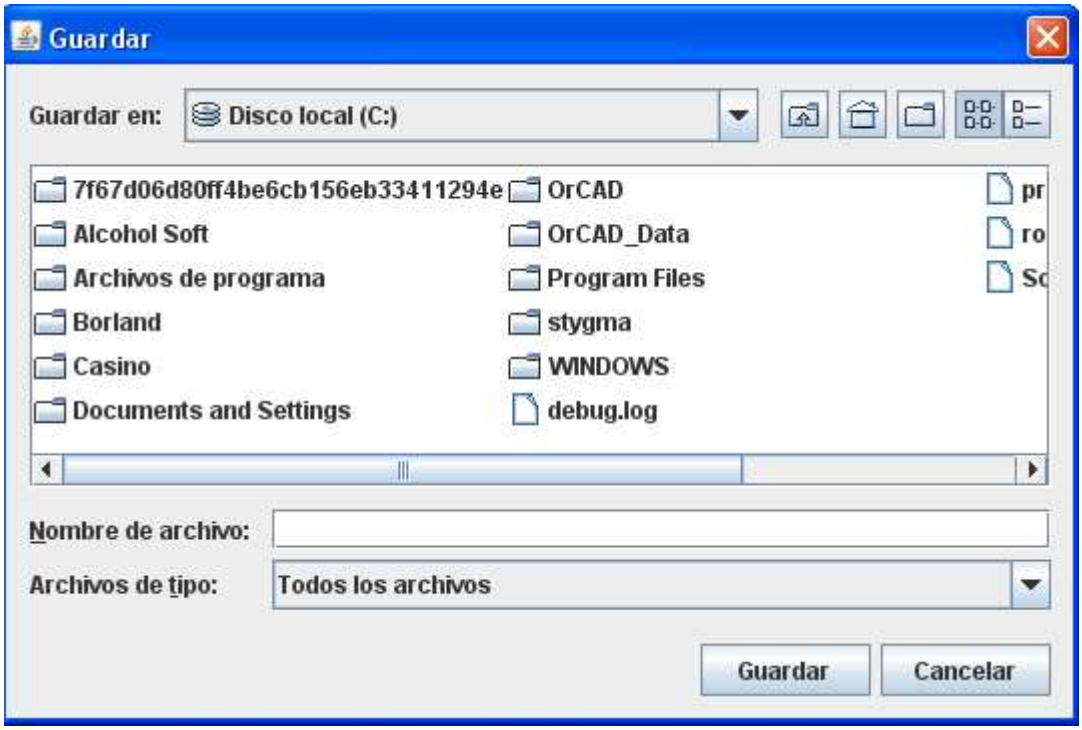


Figura 27.

-Tras elegir el directorio donde queremos guardar el archivo, y el nombre de nuestro archivo, podemos darle a cancelar para cancelar la operación de guardar, o darle a guardar y el fichero quedará automáticamente guardado y esta ventana se cerrará automáticamente.

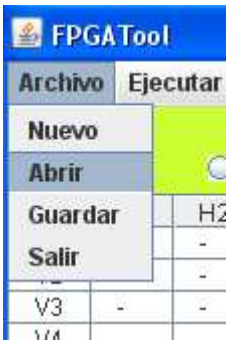


Figura 28.

-Para abrir un archivo se siguen prácticamente los mismos pasos solo que ahora pinchamos en el menú “Archivo” y luego en el submenú “Abrir” y nos aparecerá una ventana como la anterior donde elegiremos el archivo que queramos abrir. Si el archivo es abierto correctamente, inmediatamente nos aparecerá la configuración que hemos abierto tal y como la habíamos guardado.

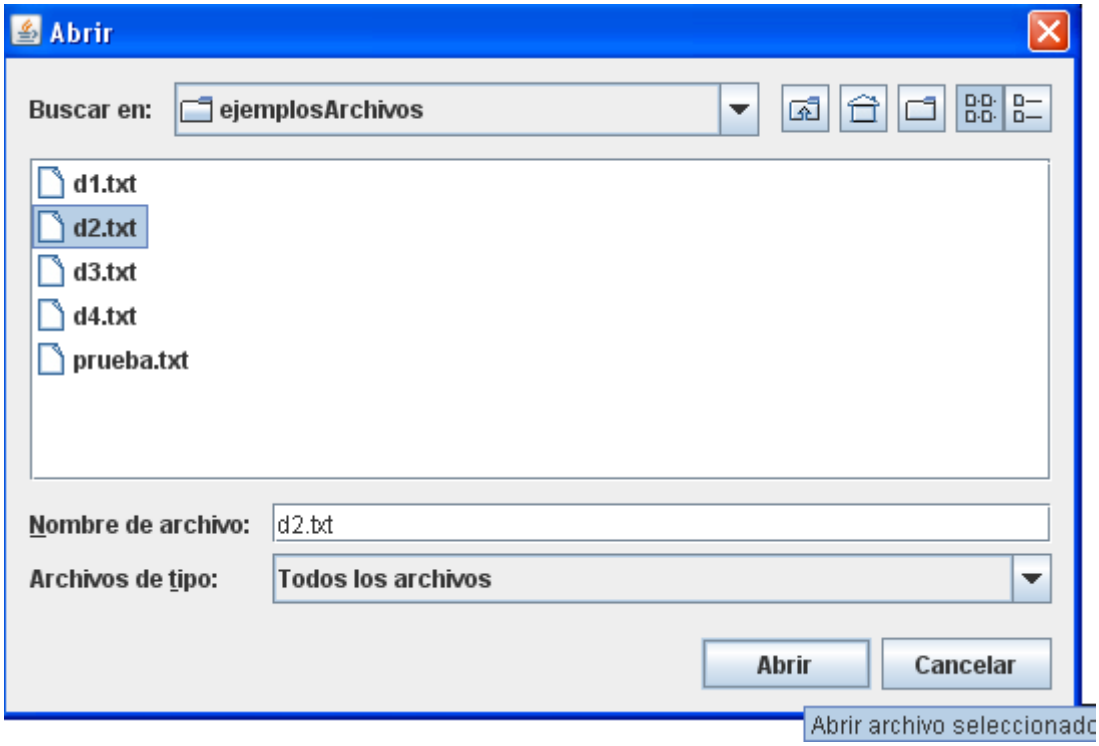


Figura 29.

5.-Simulación de la ejecución de un lote de tareas.

-Con esta herramienta tenemos la posibilidad de simular la ejecución de cualquier archivo compuesto por un lote de tareas cada una con sus características, de forma rápida y sencilla.

-Para comenzar una ejecución primero deberemos de seleccionar la técnica de ubicación de tareas que queremos que se emplee durante el transcurso de la ejecución: la forma de elegir la técnica de ubicación ya se ha explicado en el sección 3.

-Tras seleccionar una técnica de ubicación de tareas, a continuación le damos al menú "Ejecutar" tal y como muestra la siguiente imagen:

Figura 30.

-Entonces nos aparecerá una ventana con un mensaje sobre la técnica elegida para emplear en la ejecución del lote de tareas. Si no estamos de acuerdo con la configuración, podemos darle al botón de cancelar y elegir la técnica de ubicación de tareas deseada. Si por otro lado estamos de acuerdo con la configuración, entonces le damos al botón aceptar como muestra esta imagen:

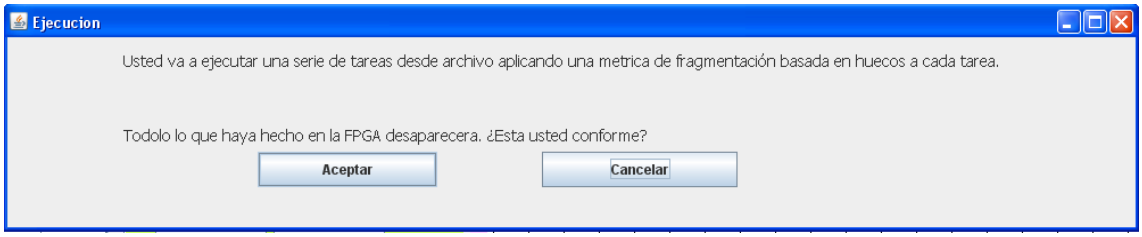


Figura 31.

-Si le damos al botón de aceptar, automáticamente desaparecerá esta ventana para dar paso a la ventana antes mostrada de abrir archivo. A continuación seleccionamos el archivo que contiene el lote de tareas que queremos ejecutar y le damos a aceptar. Si el archivo es leído correctamente, entonces se cerrará la ventana de abrir archivo y nos aparecerá una buena ventana que contiene dos botones como muestra la siguiente figura:

Figura 32.

-Ahora tenemos la opción de ejecutar evento a evento el lote de archivos o ejecutar el lote de archivos de una sola vez.

-Recordemos que un evento es el resultado de que en la FPGA entre o salga una tarea. Para ejecutar evento a evento, le daremos al botón llamado “Step”. Ejecutar evento a evento tiene la desventaja de que es más lento que si ejecutamos el lote de archivos de una sola vez. Pero tiene la ventaja de que evento a evento, podemos ver el estado de la FPGA y ver los valores correspondientes para ese estado.

-Para ejecutar el lote de tareas de una sola vez, basta con darle al botón “Run” y tras un tiempo, tendremos finalizada la simulación de la ejecución del lote de tareas.

Apéndice 2: Visualización de datos en la herramienta.

-En esta herramienta se puede visualizar de forma inmediata para un estado concreto de la FPGA, valores que nos pueden servir de gran utilidad: información en tablas de cada una de las tareas existentes y el valor de la fragmentación según una o otra medida para un estado concreto de la FPGA son algunos de los conjuntos de datos que se obtienen de forma inmediata en esta herramienta.

-Independientemente de la forma en que realicemos la ejecución de un lote de tareas, ya sea evento a evento o ejecución total, al final de la ejecución desaparecerá la ventana mostrada anteriormente para dar paso a una nueva ventana la cual contendrá información interesante sobre el resultado de la ejecución. Sin entrar en detalle por ahora, con cada uno de los datos mostrados del resultado de la ejecución, mostramos una imagen de un posible resultado de una ejecución. El valor de los datos son concretos para un lote de archivos y una técnica de ubicación, pero los datos que se muestran en cada ejecución son siempre los mismos y son los siguientes:

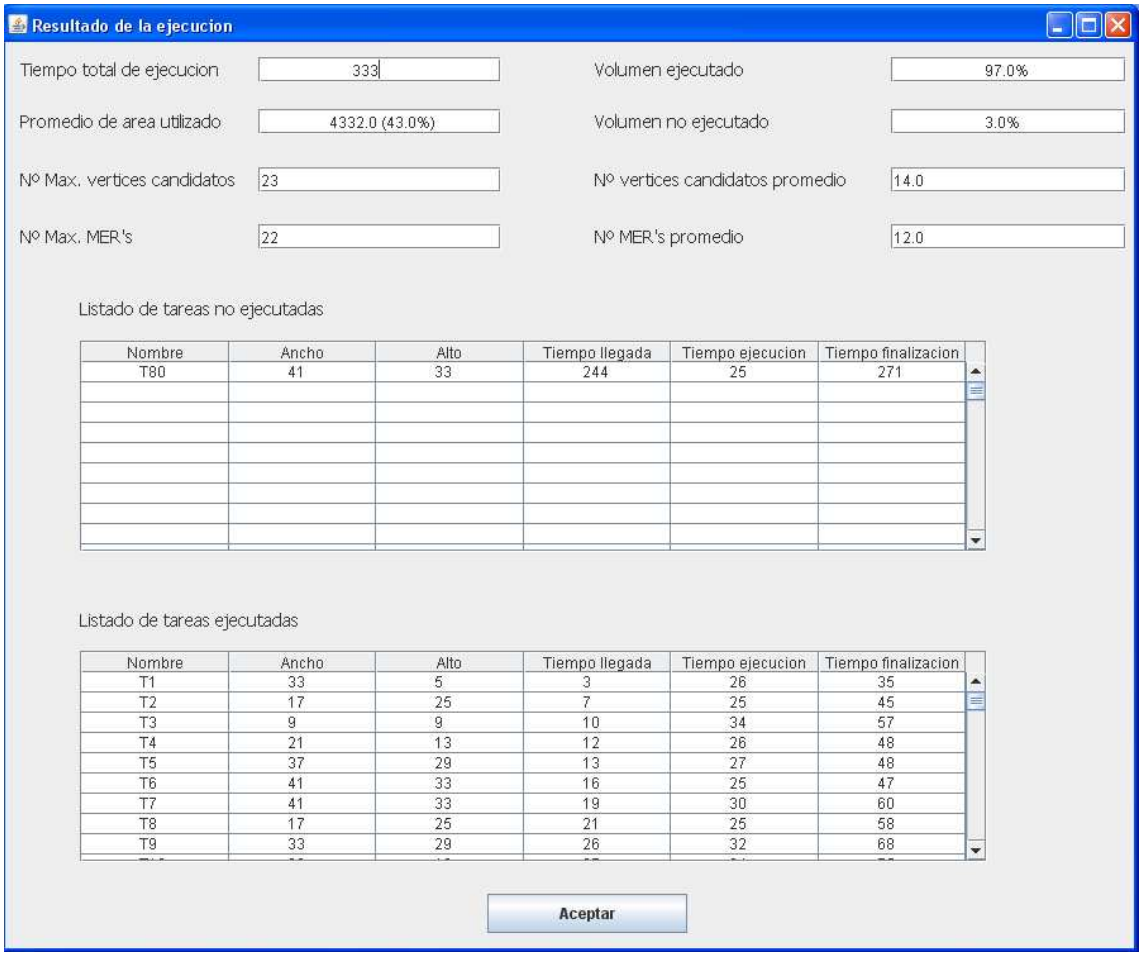


Figura 33.

-Otros datos que se pueden visualizar de forma inmediata en la interfaz de usuario son los siguientes:

-Para cualquier estado que tengamos de la FPGA, podemos visualizar directamente datos que nos pueden servir de ayuda. Estos datos aparecen por distintas zonas y se explicarán a continuación.

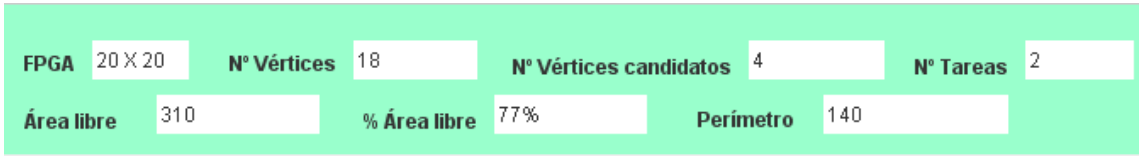


Figura 34.

-El panel que se muestra justo arriba, nos proporciona datos sobre la FPGA y son los siguientes:

-FPGA: nos da las dimensiones actuales de nuestra FPGA.

-Nº Vértices: nos dice para el estado actual de la FPGA, el número de vértices totales de las listas de vértices.

-Nº Vértices candidatos: nos da el número de vértices que tienen la propiedad de ser candidatos.

-Nº Tareas: nos indica el número de tareas que están ejecutándose en ese momento en la FPGA.

-Área libre: cantidad de área libre en la FPGA.

-% Área libre: nos indica el porcentaje de área libre de la FPGA calculado de la siguiente forma: (area_libre/area_total) * 100.

-Perímetro: es el perímetro que conforma el área libre de nuestra FPGA.

-La siguiente imagen muestra los datos correspondientes a la medida de fragmentación para el estado que tengamos de la FPGA en ese momento a partir de las dos medidas de fragmentación ya explicadas.

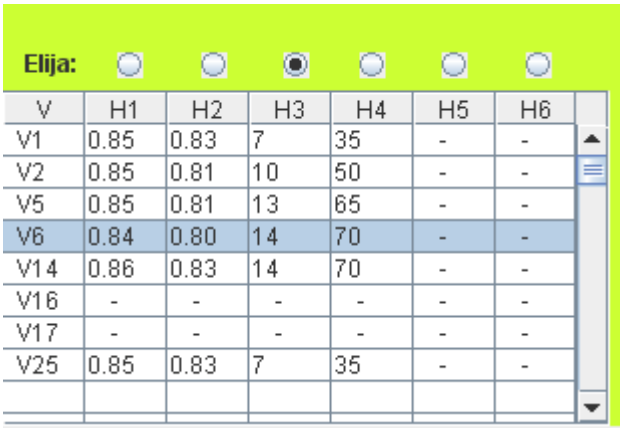
Figura 35.

-F1: muestra al valor de la medida de fragmentación basada en huecos.

-F2: muestra al valor de la medida de fragmentación basada en perímetro.

-Los demás huecos están para poder añadir alguna otra posible medida de fragmentación.

-La siguiente imagen muestra una tabla donde cada fila se corresponde con un vértice candidato y cada columna con una técnica de ubicación de tareas distinta. De forma que si estamos añadiendo una tarea a partir de una técnica de ubicación de tareas tal y como se ha explicado en la sección 3 de este manual, independientemente de qué técnica de ubicación tengamos seleccionada, en la tabla podemos ver de para cada vértice candidato y para cada técnica de ubicación, los distintos valores que se calculan.
Si para alguna fila aparecen todos los valores con el signo “-” esto quiere decir que la tarea que queremos ubicar no entra en ese vértice candidato en concreto.



V	H1	H2	H3	H4	H5	H6
V1	0.85	0.83	7	35	-	-
V2	0.85	0.81	10	50	-	-
V5	0.85	0.81	13	65	-	-
V6	0.84	0.80	14	70	-	-
V14	0.86	0.83	14	70	-	-
V16	-	-	-	-	-	-
V17	-	-	-	-	-	-
V25	0.85	0.83	7	35	-	-

Figura 36.

-La fila que está marcada en azul, es la fila que marca al vértice candidato seleccionado por la técnica de ubicación que tenemos marcada en ese momento, en este caso la heurística de adyacencia 2D.

-Con esta tabla cuando estamos incluyendo una tarea a partir de las técnicas de ubicación, podemos ver de forma global todos los resultados para cada vértice candidato y para cada técnica de ubicación.

-La imagen que viene a continuación muestra una tabla parecida a la anterior con la salvedad de que ahora cada fila representa una tarea de las que se están ejecutando en la FPGA. Las columnas indican el nombre, ancho, alto posición y tiempo restante de ejecución da cada tarea.

Nombre	Ancho	Alto	Posicion	Tiempo
T1	6	7	(7,2)	8
T2	6	8	(5,13)	7
T3	8	1	(1,1)	8
T4	8	2	(19,1)	8
T5	3	6	(15,9)	8
T6	2	2	(19,19)	8

Figura 37.

-De esta forma tenemos a nuestra disposición la información de cada una de las tareas que están ejecutándose en a FPGA, por si necesitamos algún dato de alguna tarea en concreto.

La última imagen muestra las opciones de visualización y la posibilidad de elegir cualquiera de ellas. Las opciones de simulación disponibles son las siguientes:

-FPGA: muestra o no la FPGA. Si no esta seleccionada, no se mostrará nada.

-GRID: muestra el área de la FPGA a nivel de celdas.

-ListaAristas: muestra las listas de vértices para la situación actual de la FPGA.

-MER: muestra el conjunto de MERs que se forma en el área libre de la FPGA.

-Nº Vértices: muestra el número de cada vértice. Esta información se ve si el botón de ListaAristas está seleccionado.

-Tiempo de arista: muestra el valor de cada una de las aristas. Esta información se ve si el botón de ListaAristas está seleccionado.

-Tareas: muestra o no las tareas que se están ejecutando un la FPGA en el estado actual.

-Tiempo de tarea: muestra el tiempo restante de ejecución de cada tarea. Esta información es visible si el botón Tareas está seleccionado.

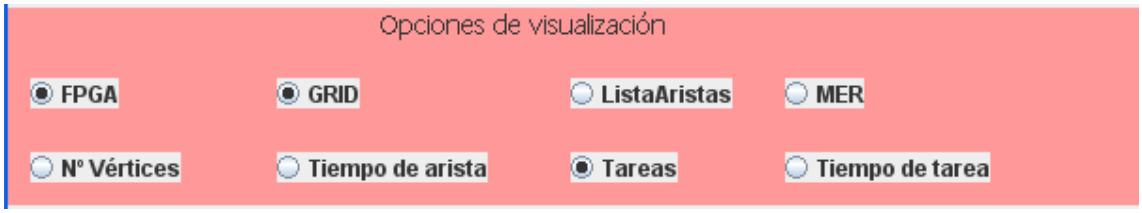


Figura 38.

Apéndice 3: Ejemplo de ejecución.

-En este apéndice se explicará cómo poder realizar la simulación de la ejecución de un archivo compuesto por un lote de tareas las cuales tienen una determinada característica.

-Lo primero de todo será explicar cómo podemos crear un archivo compuesto por un lote de tareas, ya que para que la aplicación lo pueda leer correctamente, el archivo tiene que tener un formato determinado tal y como se muestra en la siguiente figura:

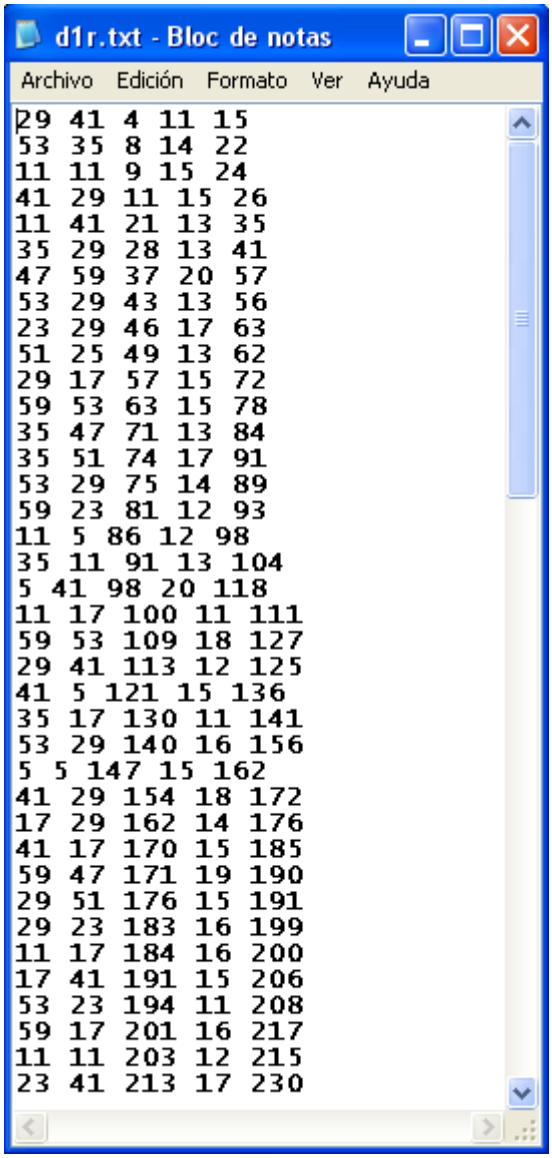


Figura 39.

-En la figura justo arriba se pueden observar distintas filas compuestas por números espaciados. Cada fila representa la información de una tarea y ya que la información de cada tarea se representa en una sola línea, pues nuestro archivo deberá tener tantas líneas como tareas queramos que se componga la ejecución.

-Si nos fijamos en cada una de las filas, aunque los números sean distintos se puede ver que siempre son cinco números, ni más ni menos. Estos cinco números son los valores de las cinco propiedades que tenemos que dar de cada tarea y que son las que a continuación vamos a enumerar por orden de izquierda a derecha según aparecen en cada línea del archivo:

-Primera columna: Indica el ancho en celdas de la FPGA de la tarea.

-Segunda columna: Indica el alto en celdas de la FPGA de la tarea.

-Tercera columna: Indica el instante de llegada de la tarea a la FPGA. El instante de llegada de la tarea no quiere decir que sea en el instante en el que la tarea pase a ejecución, ya que si no hay área libre suficiente en ese momento, pues la tarea tendrá que esperar hasta que haya espacio suficiente como para que pueda ejecutarse.

-Cuarta columna: Indica el tiempo que la tarea tiene que estar ejecutándose en la FPGA para verse completada.

-Quinta columna: nos indica el time-out o tiempo máximo de finalización que tiene la tarea. Las tareas tienen la propiedad de que deben haber sido ejecutadas en un plazo, si la tarea va a ejecutarse en un instante X y su tiempo de ejecución es de Y, pues antes de entrar a ejecutarse, se comprueba que X + Y sea menor o igual que su time-out, ya que si no lo es, es que la tarea se terminaría ejecutando fuera de plazo, por lo tanto si ocurre esta situación, la tarea será descartada.

-De esta forma conseguimos tener un archivo con tantas tareas como se quiera listo para ser simulado en la aplicación. Donde cada tarea viene perfectamente determinada por sus cinco propiedades que son T = {ancho, alto, tiempo_llegada, tiempo_ejecucion, time-out}.

-Sobre el formato del archivo, decir que éste sólo puede tener tantas filas como tareas queramos que haya, ni una fila más ni una fila menos, y que cada tarea venga en una única fila. Esto da lugar a que el archivo no puede llevar ninguna otra información que no sea la información de cada tarea, por lo tanto este formato tampoco admite comentarios de ningún tipo.

-Quedando claro el formato del archivo, sólo falta decir que tal archivo tendrá que ser creado con el nombre que se desee pero con la salvedad de que sea en formato de texto, es decir, de tipo .txt. Se ha seleccionado este tipo de archivo y no de tipo binario por ejemplo, para poder comprobar visualmente las tareas y cada una de las características simplemente abriendo el archivo, ya que viene en formato de texto.

-Ahora queda que queda claro cómo crear un archivo formado por un conjunto de tareas con sus características para simular su ejecución, vamos a ir viendo paso a paso cómo evoluciona la ejecución de un archivo de tareas en concreto, explicando todas las posibilidades que nos permite la herramienta a la hora de ejecutar.

-A continuación se explica cómo llevar a cabo la ejecución de un lote de tareas detallando cada uno de los pasos que hay que seguir. Ahora simplemente se enumeran a modo de algoritmo:
Algoritmo de ejecución.

-Paso 1: Crear un archivo formado por un lote de tareas con el formato antes explicado, nombrarlo y guardarlo con el tipo .txt.

-Paso 2: Abrir la aplicación si es que no estaba abierta.

-Paso 3: Seleccionar la técnica de ubicación de tareas que queremos que utilice la herramienta para colocar cada una de las tareas del archivo a lo largo de la ejecución.

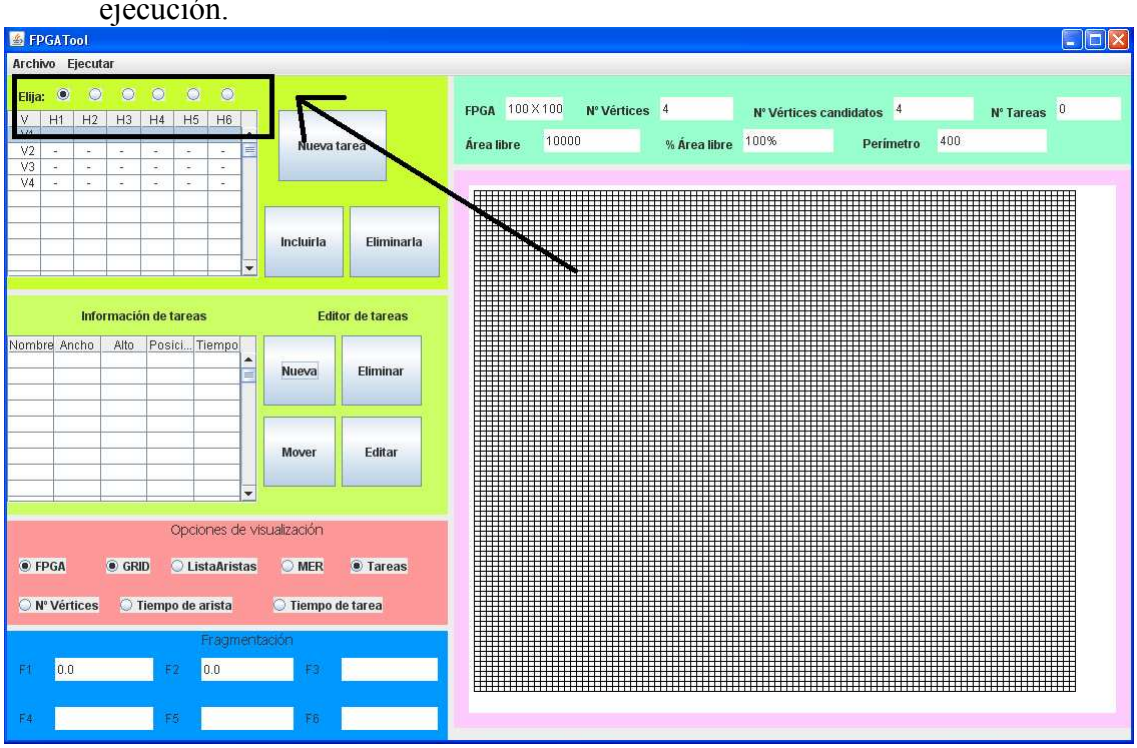


Figura 40.

-Paso 4: Tras hacer clic sobre el menú “Ejecutar”, tenemos que revisar si la información que nos aparece en una ventana sobre la ejecución es correcta. Si no es correcta, basta con darle al botón de cancelar que aparece en la ventana y volvemos al paso 3. Si por el contrario la información es correcta, le damos al botón aceptar que aparece en la ventana y avanzamos al paso siguiente.

-Paso 5: Seleccionamos el archivo que hemos creado anteriormente u otro archivo cualquiera sierre que tenga el formato correcto. Si nos arrepentimos de realizar la ejecución, basta con darle al botón de cancelar que aparece en el cuadro de diálogo para abrir el fichero. Si por el contrario queremos llevar a cabo la ejecución de forma definitiva, le damos al botón de abrir que aparece en el cuadro de diálogo para abrir el fichero o hacemos doble click con nuestro ratón en el archivo que queremos abrir e inmediatamente comenzaremos la ejecución.

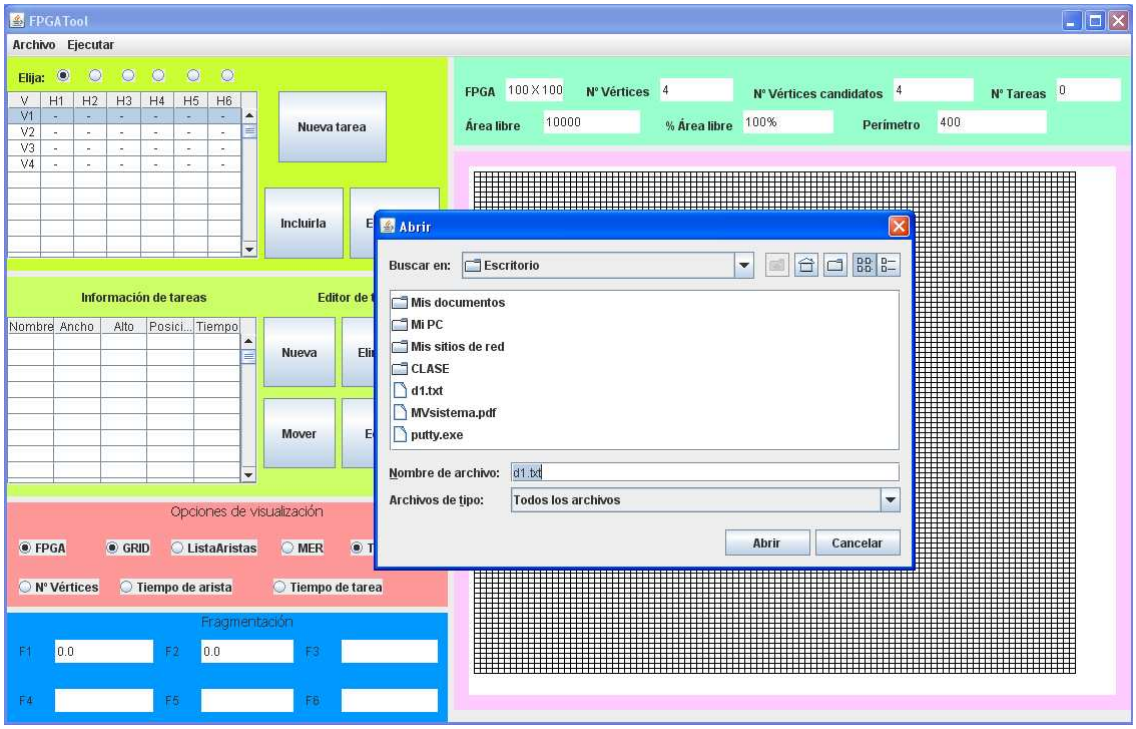


Figura 41.

-Paso 6: Ahora nos aparece una ventanita de ejecución con los botones “step” y “run”, ambos sirven para ejecutar el lote de tareas contenido en el archivo. La diferencia es que presionando el botón “step”, haremos que la ejecución avanze hasta que en la FPGA ocurra un evento. Por otro lado si hacemos clic obre el botón “run”, la ejecución se llevará a cabo hasta que todas las tareas que se hayan podido ejecutar sean ejecutadas por completo. Si le damos al botón “run” nos vamos al paso siguiente. Si por el contrario le damos al botón “step”, de nuevo repetimos el paso 6 a no ser que la ejecución haya finalizado, y en este caso iremos también al paso siguiente.

-Paso 7: En este paso estamos en la situación en la que la ejecución ha llegado a su fin y se nos muestra una ventana con información referente a la ejecución. Tras presionar el botón aceptar de esta nueva ventana, la ejecución habrá terminado.

-Siguiendo este algoritmo, se mostrará a continuación un ejemplo de ejecución concreto con sus respectivas imágenes. En este ejemplo en concreto daremos primero varios pasos de ejecución evento a evento para así poder ir viendo cómo se van colocando las tareas que se ejecutan en la FPGA. Tras unos pasos, correremos la ejecución hasta que finalice dándole al botón “run”.

-Ejemplo de ejecución siguiendo el algoritmo:

-Paso 1: creamos el siguiente archivo de ejecución siguiendo el formato establecido. A continuación se muestran capturas del archivo en concreto.

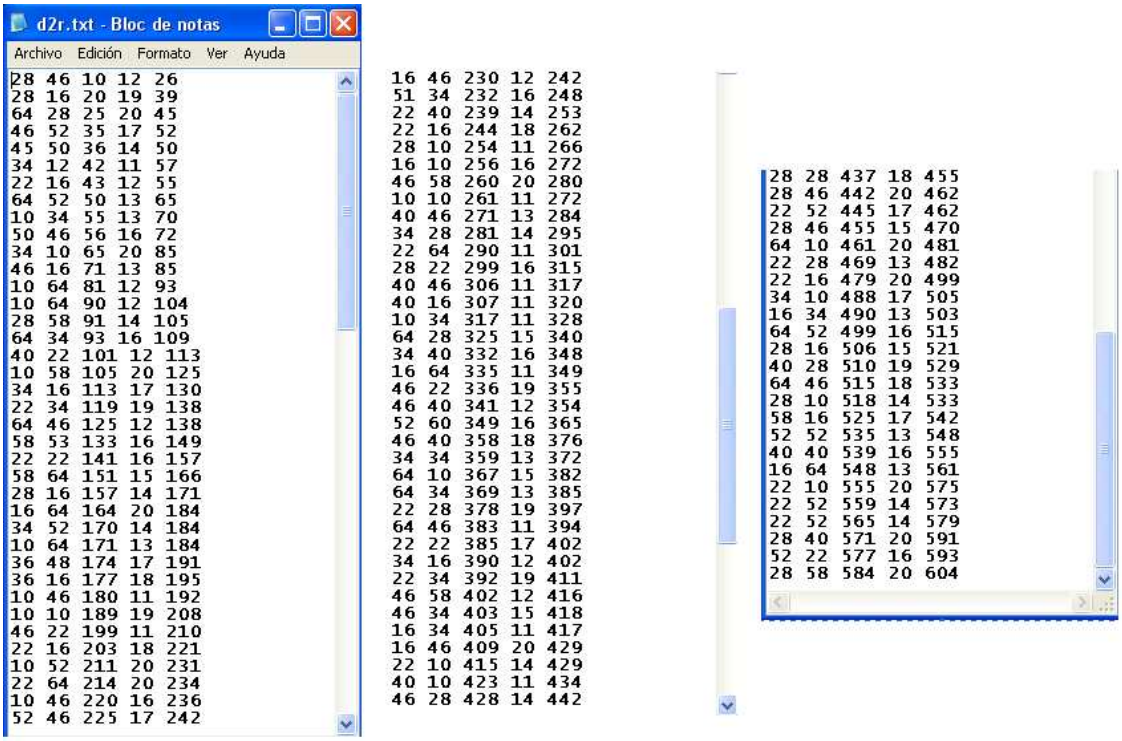


Figura 42.

-Paso 2: Abrimos la aplicación.

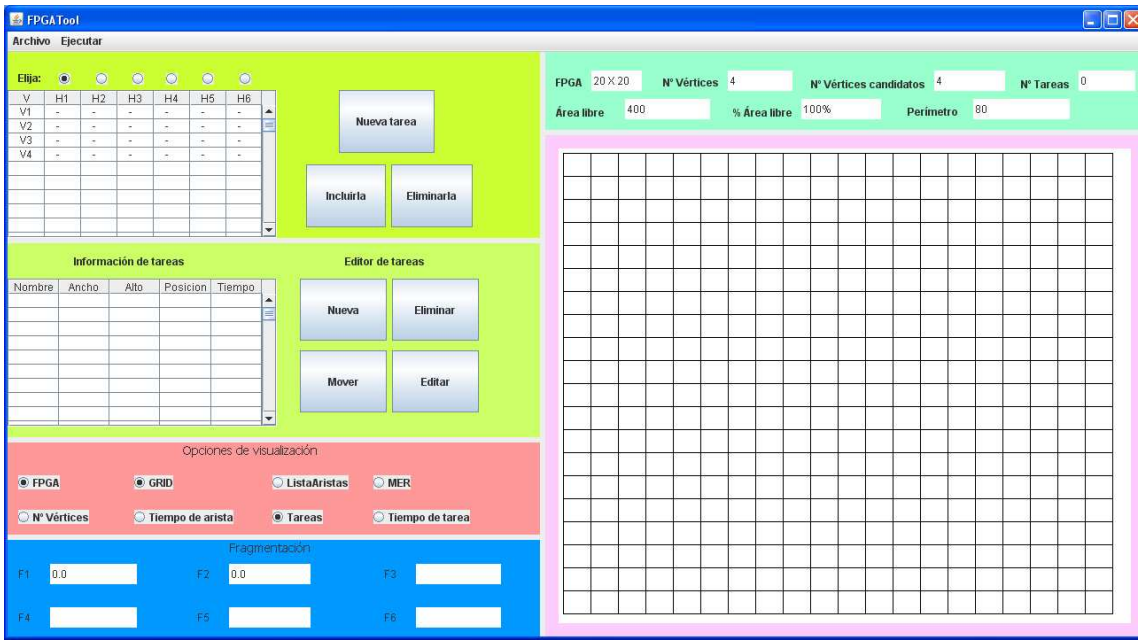


Figura 43.

-Paso 3: Elegimos como técnica de ubicación la basada en la heurística de adyacencia 3D que se corresponde con el nombre H4 tal y como se explica en el apéndice 1.

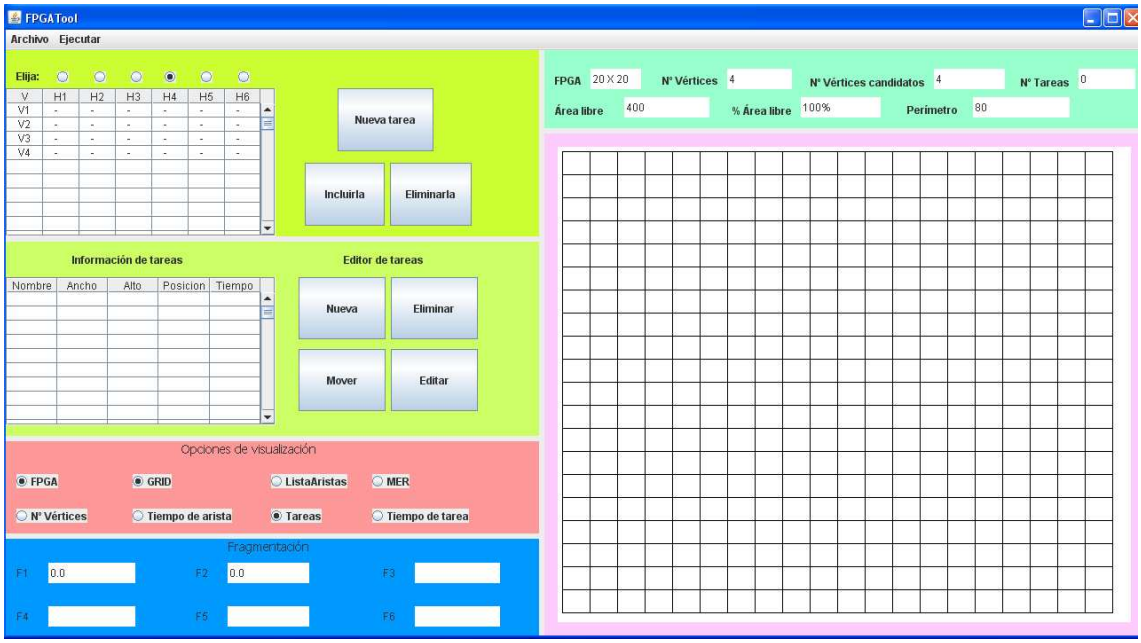


Figura 44.

-Paso 4: Hacemos clic sobre el menú "Ejecutar" en la parte superior izquierda de la aplicación. Comprobamos que la información sobre la ejecución es correcta y le damos al botón de aceptar.

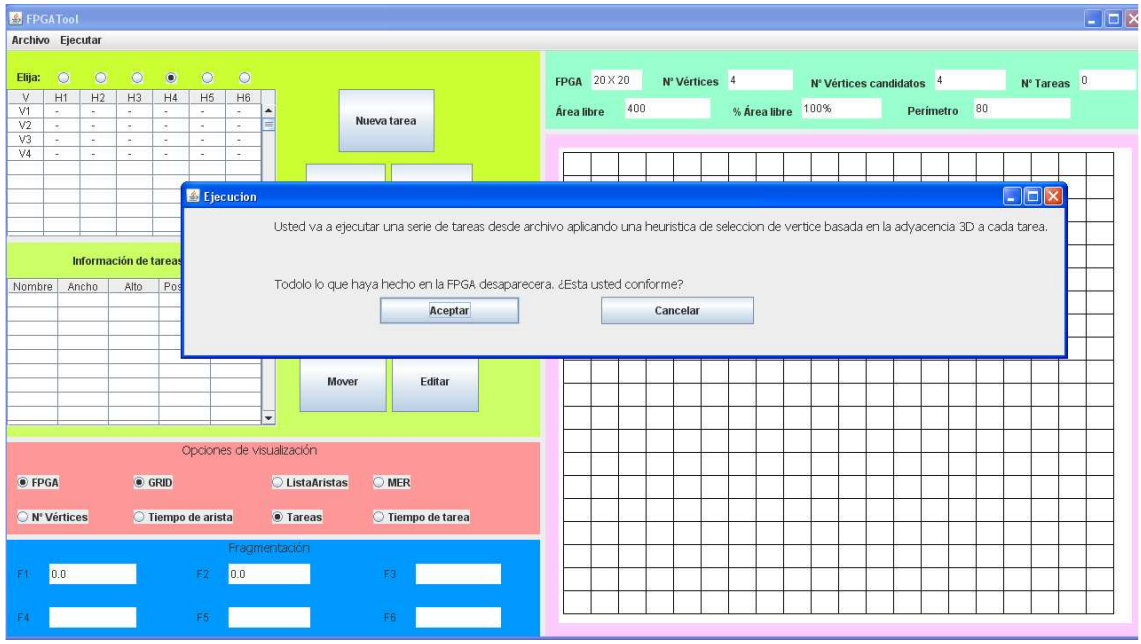


Figura 45.

-Paso 5: Buscamos el archivo d2r.txt y le damos al botón de abrir.

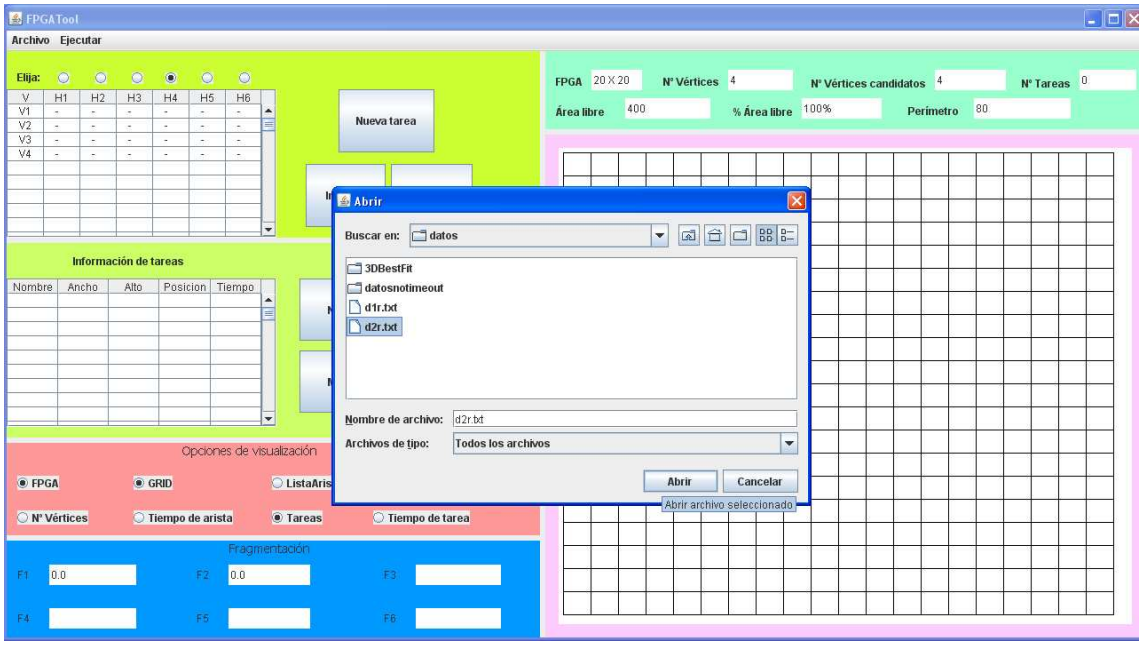


Figura 46.

-Paso 6(1): Nos aparece la ventana de ejecución y elegimos ejecutar hasta un próximo evento haciendo clic sobre el botón “Step”.

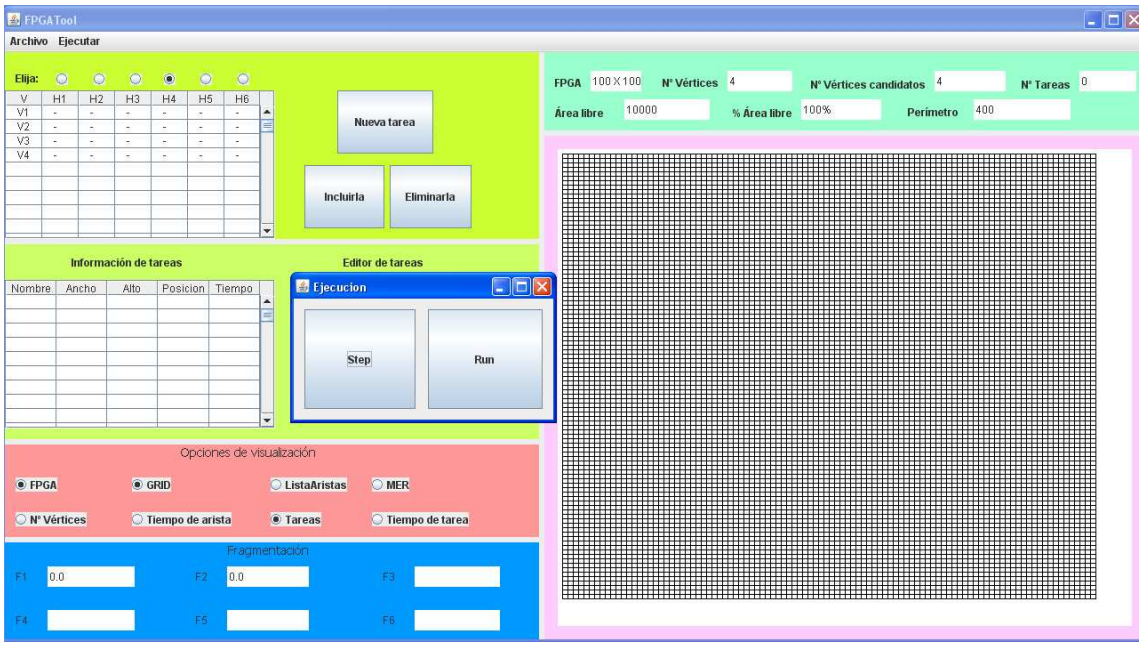


Figura 47.

-Ahora se muestra el resultado de la ejecución tras el primer evento, permitiéndonos la herramienta poder visualizar el conjunto de MERs o las listas de vértices. En este paso visualizaremos los MERs.

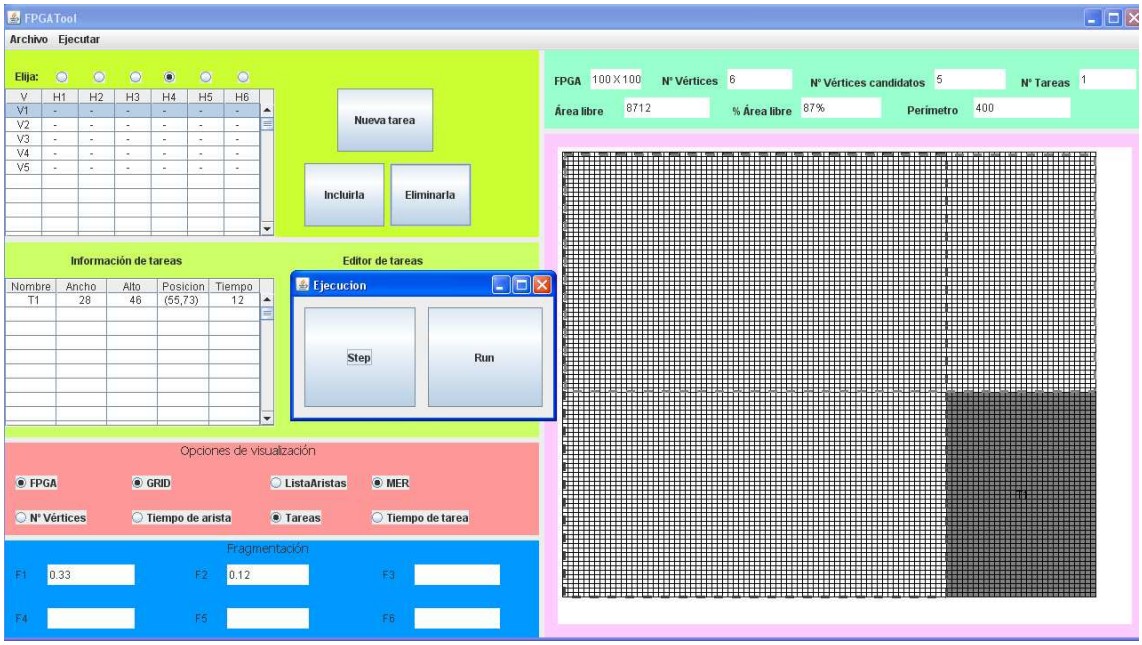


Figura 48.

-Paso 6(2): Queremos visualizar la ejecución hasta un nuevo evento, por lo tanto le damos de nuevo al botón “Step” y esta vez visualizaremos las listas de vértices sin ninguna información sobre éstas.

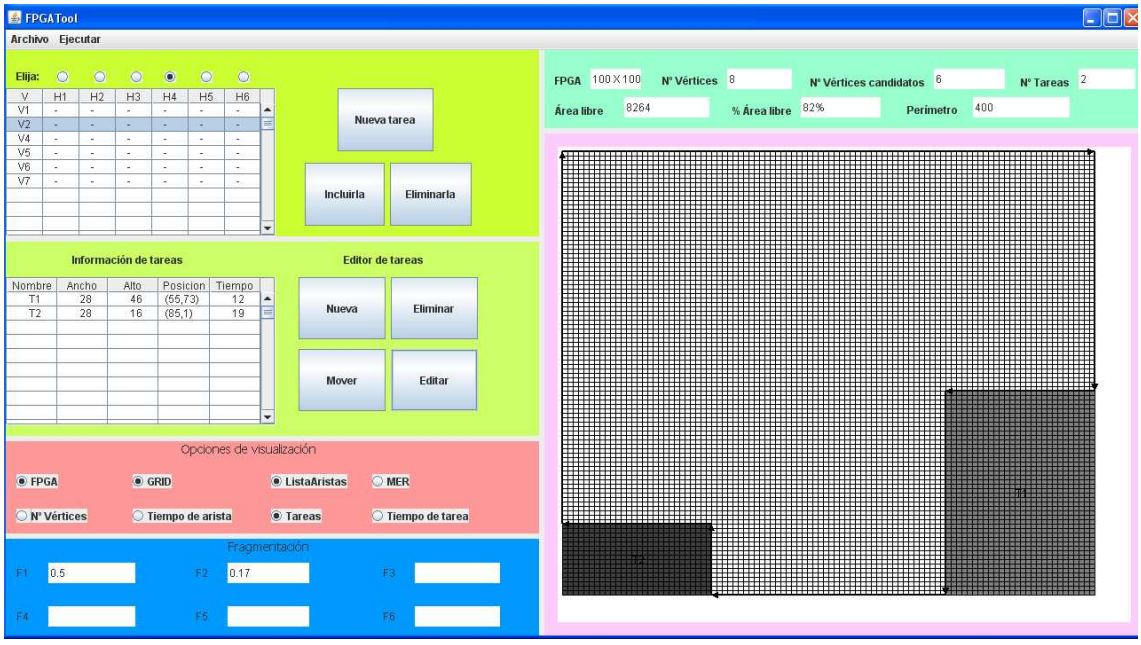


Figura 49.

-Paso 6(3): De nuevo le damos al botón “Step” para visualizar el estado de la ejecución hasta el siguiente evento, mostrando ahora tanto las listas de vértices

con el valor de cada arista y con los vértices enumerados, como el conjunto de MERs.

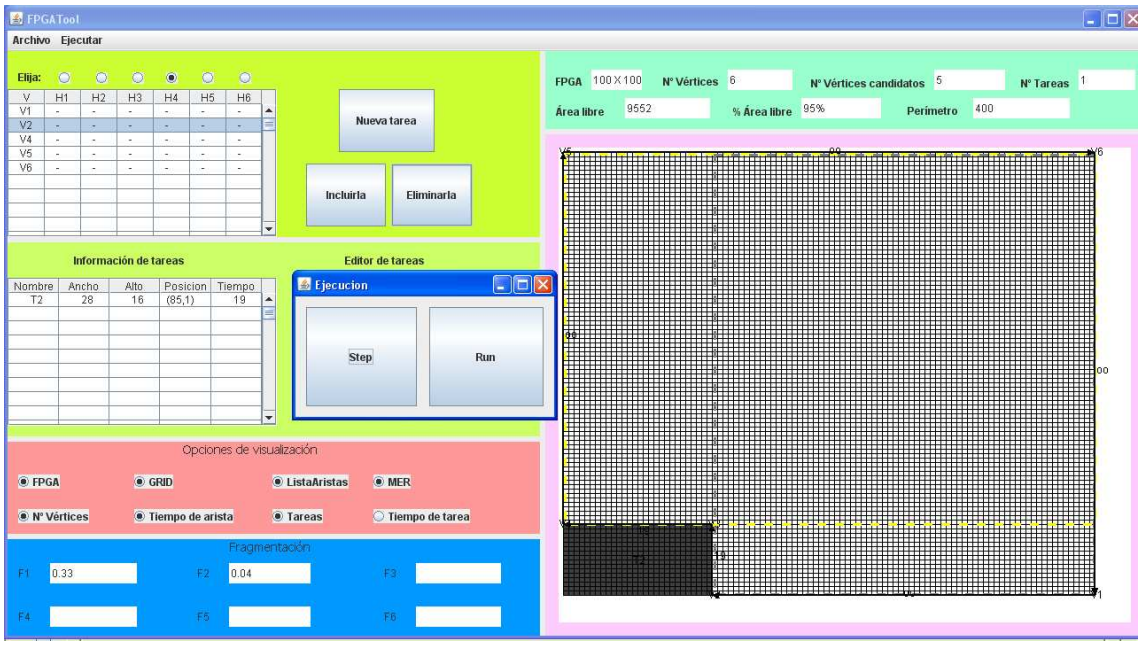


Figura 50.

-En este paso podemos observar que la tarea T1 ha sido ejecutada y por lo tanto desaparece de la FPGA, mientras que la tarea T2 sigue ejecutándose.

-Paso 6(4): Ahora queremos que la ejecución se complete de una sola vez ya que lo que nos interesa es obtener los datos finales de la ejecución. Entonces le damos al botón “Run” y tras un tiempo, la ejecución se habrá realizado por completo.

-Paso 7: Automáticamente tras la finalización de la ejecución del lote de tareas en la FPGA, se nos muestra una ventana con una serie de datos que se explicarán a continuación:

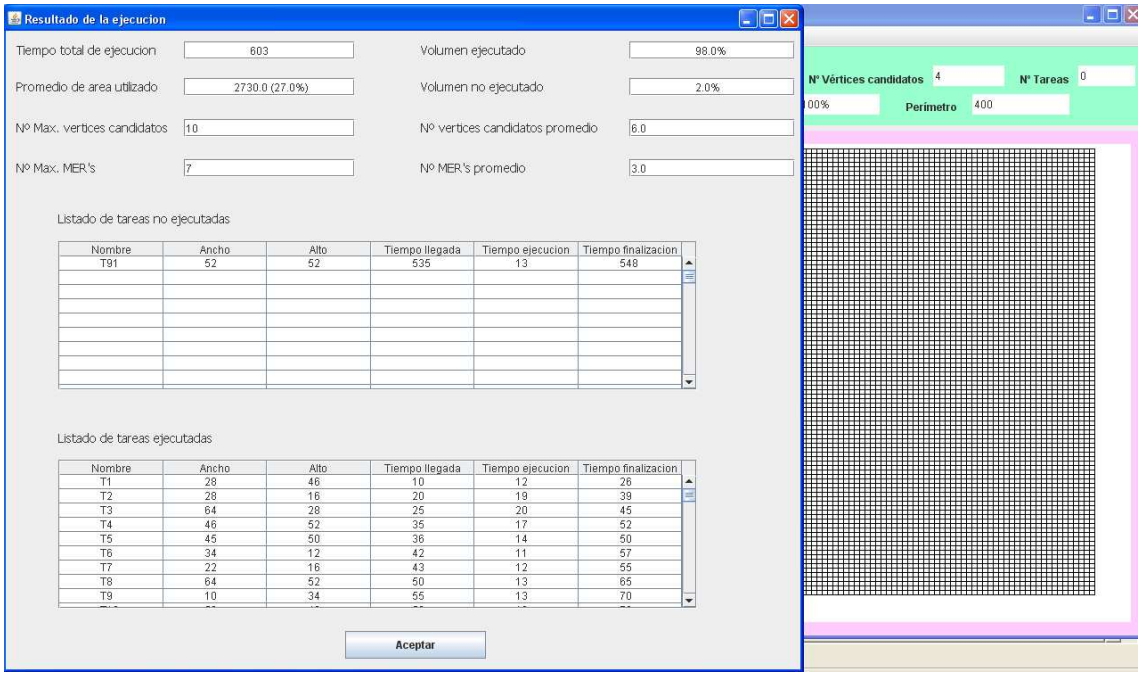


Figura 51.

-Los datos que se muestran en la ventana de resultados de la ejecución son los siguientes:

-"Tiempo total de ejecución": es el tiempo que ha tardado la FPGA en realizar la ejecución del lote de tareas.

-"Promedio de área utilizado": aquí nos aparecen dos valores, un número y un tanto por ciento. El número hace referencia a la cantidad de celdas ocupadas por tareas durante el periodo de ejecución en promedio. El porcentaje indica el tanto por ciento de celdas ocupadas por tareas en relación al número de celdas totales de la FPGA en promedio durante el periodo de ejecución. Este porcentaje nos indica el tanto por ciento en promedio que la FPGA ha sido aprovechada.

-"Nº Max. MERs": nos dice cuántos MERs ha habido en el evento con mayor número de MERs de entre todos los eventos que han ocurrido en la FGPA durante la ejecución.

-"Nº MERs promedio": es la suma del número de MERs de cada evento a lo largo de la ejecución dividido entre el número de eventos.

-"Nº Max. vertices candidatos": nos dice cuántos vértices candidatos ha habido en el evento con mayor número de vértices candidatos de entre todos los eventos que han ocurrido en la FGPA durante la ejecución.

-"Nº vertices candidatos promedio": es la suma del número de vértices candidatos de cada evento a lo largo de la ejecución dividido entre el número de eventos.

-“Volumen ejecutado”: se entiende por volumen de una tarea como al área que ocupa esa tarea, multiplicado por el tiempo de ejecución de esa tarea. Dicho esto, el volumen ejecutado es la suma del volumen de las tareas que se han ejecutado con relación a la suma del volumen de todas las tareas que comprenden el lote completo de tareas, ya que es probable que ciertas tareas no hayan podido ser ejecutadas.

-“Volumen no ejecutado”: el volumen no ejecutado es la suma del volumen de las tareas que no se han ejecutado con relación a la suma del volumen de todas las tareas que comprenden el lote completo de tareas.

-“Listado de tareas no ejecutadas”: como su nombre indica, es un listado de todas las tareas que no se han podido ejecutar. La información se muestra en una tabla en donde cada fila corresponde a una tarea que no se ha ejecutado con su información correspondiente:

-“Nombre”: nombre de la tarea.
-“Ancho”: ancho de la tarea en número de celdas.
-“Alto”: alto de la tarea en número de celdas.
-“Tiempo de llegada”: el tiempo de llegada a la FPGA de la tarea.
-“Tiempo de ejecucion”: tiempo de ejecución de la tarea.
-“Tiempo finalizacion”: tiempo máximo de que tiene la tarea para ser ejecutada.

-Tras obtener la información que nos interese de los datos mostrados, le damos al botón de aceptar y la ejecución habrá terminado, dejándonos por defecto un estado de la FPGA vacía de tamaño 100 X 100.

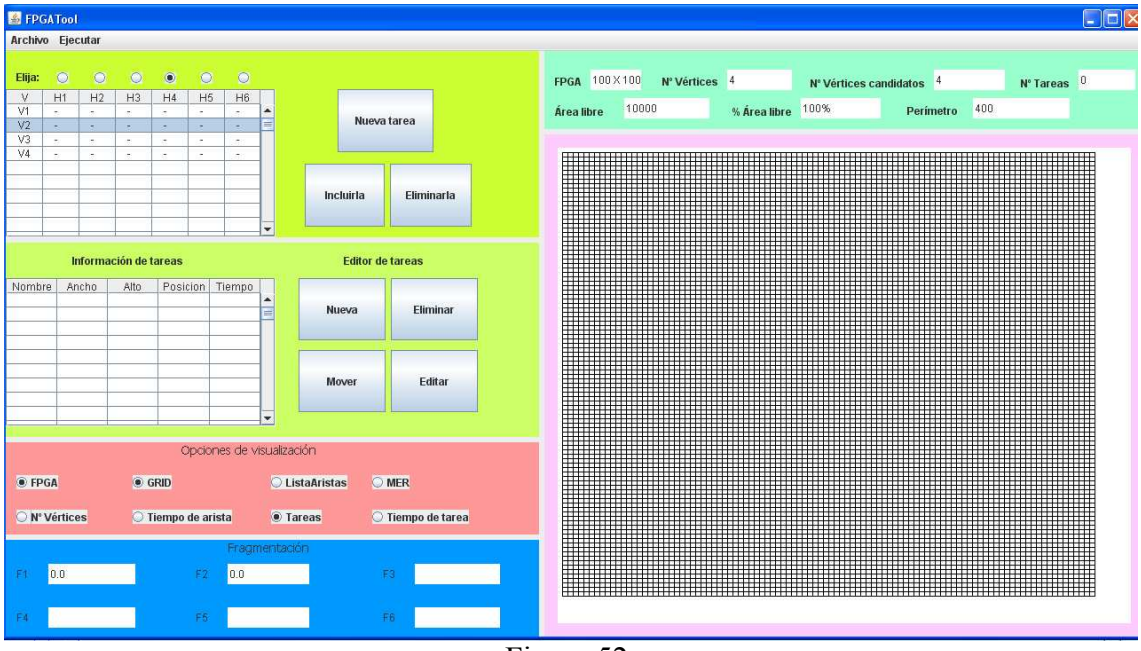


Figura 52.

Referencias

[1] Xilinx, Inc "Virtex Datasheet", DS003, http://www.xilinx.com,.

[2] Xilinx, Inc. "Virtex-4 Configuration Guide", UG071, http://www.xilinx.com.

[3] Xilinx, Inc. "Virtex-5 Configuration User Guide", UG191, http://www.xilinx.com.

[4] Tabero, J., Septién, J., Mecha, H., Mozos, D., "Task Placement Heuristic Based on 3-D Adjacency and Look-Ahead in Reconfigurable Systems", Proc. of the 11th Asia and South Pacific Design Automation Conference, ASP-DAC 2006, Yokohama, Japan, pp. 396-401.

[5] Septién, J., Mecha, H., Mozos, D., Tabero, J., "2D Defragmentation Heuristics for Hardware Multitasking on Reconfigurable Devices", Proc. of the 13th Reconfigurable Architectures Workshop, RAW'06, Proc. Int. Parallel and Distributed Processing Symposium, Greece, April 2006.

[6] Tabero, J., Septien, J., Mecha, H., Mozos, D., "A Low Fragmentation Heuristic for Task Placement in 2D RTR HW Management", Proc. Int. Conf. on Field-Programmable Logic and Application, Lecture Notes in Computer Science, vol. 3203. Springer-Verlag, 2004, pp. 241-250.

[7] Jin Cui, Zonghua Gu, Weichen Liu and Qingxu Deng, "An Efficient Algorithm for Online Soft Real-Time Task Placement on Reconfigurable Hardware Devices", IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 07), Mayo, 2007.

[8] K. Bazargan, R. Kastner, M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems", IEEE Design and Test of Computers, volume 17, pg. 68–83, 2000.